January 2016

Graph diffusions and matrix functions: fast algorithms and localization results

Kyle Kloster
Purdue University

Graduate School Form
30 Updated 12/26/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Kyle Kloster

Entitled

GRAPH DIFFUSIONS AND MATRIX FUNCTIONS: FAST ALGORITHMS AND LOCALIZATION RESULTS

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

David F. Gleich
Co-chair

Jianlin Xia
Co-chair

Greg Buzzard

Jie Shen

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): David F. Gleich

Approved by: Greg Buzzard 4/25/2016

Head of the Departmental Graduate Program Date

GRAPH DIFFUSIONS AND MATRIX FUNCTIONS: FAST ALGORITHMS AND LOCALIZATION RESULTS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Kyle Kloster

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2016

Purdue University

West Lafayette, Indiana

For my big brother,

who first taught me what a function is,

who first made me think math looked cool,

and who has always supported me.

ACKNOWLEDGMENTS

Many teachers prepared me for this thesis. My high school math teacher George Mills had an infectious curiosity, and it was his ability to share the creative aspects of math that first set me on this path. At Fordham University, Theresa Girardi and Melkana Brakalova devoted a lot of their time to guiding me along, and Gregory V. Bard has remained a mentor of mine well beyond the call of his undergraduate advisor duties. Steve Bell, who taught me the miracles of Complex Analysis, encouraged me through difficult times as a graduate student. Without any one of them, I might not have reached this finish line.

A number of staff made my stay at Purdue loads easier. Rebecca Lank, Terry Kepner, and Betty Gick steered me the right way at every turn to ensure I could graduate. Marshay Jolly, Susan Deno, and Jennifer Deno helped me through the paperwork of all my academic travels. All of them were always friendly and made my time so much more pleasant.

I had the pleasure of a very helpful research group - I want to thank Yangyang Hou, Huda Nassar, Nicole Eikmeier, Nate Veldt, Bryan Rainey, Yanfei Ren, Varun Vasudevan, and Tao Wu for their showing me new ideas, and for helpful feedback on much of the work that ended up in this thesis. Thanks to Eddie Price, Ellen Weld, and Michael Kaminski for help in confirming parts of a proof. And thanks to Tania Bahkos, Austin Benson, Anil Damle, Katie Driggs-Campbell, Alicia Klinvex, Christine Klymko, Victor Minden, Joel Pfeiffer, Olivia Simpson, Jimmy Vogel, Yao Zhu, and the entire Gene Golub SIAM Summer School classes of 2013 and 2015 for many nice conversations on research, career, and grad student life.

My co-authors, for their perseverance across time and space and delayed emails, and explaining ideas to me myriad times: Yangyang Hou, Tanmoy Chakraborty, Ayushi Dalmia, David Imberti, Yangyang Hou, Yixuan Li, Stephen Kelley, Huda Nassar, Olivia Simpson, and Merrielle Spain. I've immensely enjoyed our works together.

Finally, a number of individuals for various kinds of support: My parents, brother, and sister for always believing in me and making me laugh. Laura Bofferding and little Isaac, for sharing my advisor for so many meetings, even during Isaac's play time. My co-serving graduate representatives, Katia Vogt Geisse, Vita Kala, and especially Mariana Smit Vega Garcia for running the math department for a whole year together. Britain Cox, Jason Lucas, and Arnold Yim, for being good roommates and frisbee teammates through stressful thesis and job-search moments. Andrew Homan, Paul Kepley, Christina Lorenzo, Brittney Miller, Mike Perlmutter, Anthony and Erin Rizzie, and Kyle "Soncky" Sinclair for coursework help and grad school survival skills. Will Cerbone and Nick Thibideau, for

their endless supportive humor, and Timo Kim, for always motivating me to see what I am capable of.

David Imberti, for endless mathematical creativity, encouragement, and humorous but mathematically relevant GIFs. I haven't met anyone whose curiosity and drive for exploring and explaining phenomena better deserve the title "scientist". David was my advisor before I had an advisor.

Finally, my PhD advisor David F. Gleich, whose guidance has been as irreplaceable as it has been plentiful. When we started collaborating, at least three professors informed me I had "chosen the best advisor there is", and in the three years I have worked with him since, I never found a reason to disagree. David tirelessly introduced me to new collaborators, peers, and mentors, helped me with proofs, programming, writing, and teaching, and backed the Student Numerical Linear Algebra seminar series (PUNLAG) across my two year stint as its organizer. He also funded me endlessly, causing all my papers to contain the incantation "this work was supported by NSF CAREER Award CCF-1149756."

Bonus Acknowledgments My thesis work had a soundtrack, and it would feel incomplete not to mention the following albums that accompanied a few of the more significant proofs and programming sessions: The Flaming Lips - Yoshimi Battles the Pink Robots; Grandaddy - The Sophtware Slump, Sumday; Daft Punk - complete discography, but especially Random Access Memories; Radiohead - complete discography, but especially OK Computer and Kid A; Enya - Greatest Hits; and the soundtracks to Adventure Time, FF VI, FF VII, and Chrono Trigger.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Kloster, Kyle PhD, Purdue University, May 2016. Graph Diffusions and Matrix Functions: Fast Algorithms and Localization Results. Major Professor: David F. Gleich.

Network analysis provides tools for addressing fundamental applications in graphs such as webpage ranking, protein-function prediction, and product categorization and recommendation. As real-world networks grow to have millions of nodes and billions of edges, the scalability of network analysis algorithms becomes increasingly important. Whereas many standard graph algorithms rely on matrix-vector operations that require exploring the entire graph, this thesis is concerned with graph algorithms that are local (that explore only the graph region near the nodes of interest) as well as the localized behavior of global algorithms. We prove that two well-studied matrix functions for graph analysis, PageRank and the matrix exponential, stay localized on networks that have a skewed degree sequence related to the power-law degree distribution common to many real-world networks. Our results give the first theoretical explanation of a localization phenomenon that has long been observed in real-world networks. We prove our novel method for the matrix exponential converges in sublinear work on graphs with the specified degree sequence, and we adapt our method to produce the first deterministic algorithm for computing the related heat kernel diffusion in constant-time. Finally, we generalize this framework to compute any graph diffusion in constant time.

1. INTRODUCTION

As huge graphs like the internet and social networks have become pervasive, the ability to analyze these graphs rapidly has become important. One common approach to rapid network analysis is to focus on targeted information, i.e., to focus on studying a small set of users instead of attempt to analyze the entire network. One successful methodology for this personalized analysis is to probe the graph with a diffusive process. Imagine "heating up" the user that you are interested in studying and watching as the heat flows or diffuses to the rest of the graph. The areas of the graph where the most heat settles are then of greatest importance or relevance to the targeted user. This general problem of determining graph nodes (users) important to a target node includes such tasks as community detection, link prediction, node centrality, node similarity, and more that are used as fundamental subroutines when mining graphs for information or performing machine learning tasks.

The huge size of modern graph datasets poses a problem for even these personalized methods. Technically, such diffusive processes will always spread across an entire network if it is connected, and so exactly computing even these personalized diffusions will require looking at the whole network. The prevalence of massive datasets, especially in relation to the explosion of social media and user-personalized services, then makes these computations impractical. We need algorithms that are sublinear in the size of the graphs. Toward addressing these issues, in this thesis I present my work on developing new algorithms that address open questions in the class of *local algorithms* for network analysis.

A local algorithm is one that analyzes only a piece of the input object and requires work that depends on the size of the output rather than the size of the whole input. If good local algorithms are available, characterizing small, targeted areas of a graph or a matrix can then be done in a sublinear or even constant amount of computational effort. It is equally important to understand how the *structural properties of the graph* tie into this kind of local behavior. For example, local algorithms can become inefficient on networks that are too highly-connected, and the presence of dense subsets can cause huge slow-downs for both local and global algorithms, making it vital to understand the structural layout of a graph so that such obstacles can be intelligently avoided. This thesis describes my contributions, with my advisor professor David Gleich and other collaborators, in designing new local algorithms for popular graph diffusions and functions of matrices (such as the personalized PageRank and heat kernel vectors) as well as in proving theoretical relationships between network structures and the efficiency of algorithmic tools used to analyze those networks.

Our work focuses on computations with two categories of accuracy, which we describe intuitively here. For some applications, like community detection, the graph information that is most useful is simply the set of nodes that are most relevant to the target node; for such applications, it suffices to compute diffusions with a very low degree of accuracy (discussed in detail in Chapter 7). In the analogy of watching heat diffuse across a graph from a node you are interested in, this corresponds to the idea that you would just want to know which nodes heated up most significantly (rather than compute the exact temperature increase in any of the nodes). Algorithms that can run on a graph in a local manner to obtain this kind of lax accuracy we say are *weakly local* or have weak localization. Other applications, like ranking webpages for search returns, benefit from more refined information: when ranking webpages it is not enough to know simply that certain sites are important – you also need to put the important sites in order with some precision. For such applications we need to understand when we can compute diffusions as accurately as desired without having to look at the entire graph. (This type of accuracy is discussed rigorously in Chapter 3.) Algorithms that can obtain this stronger type of accuracy by running in a local manner we say are *strongly local* or have strong localization.

There are three types of diffusions that we consider. The first is called PageRank and corresponds to heat diffusing through a network and slowly leaking out, consistently, as the heat is flowing around the network. The second is called the heat kernel. It models a closer approximation of how heat would actually flow in the network based on thermodynamic principles. The third is a general diffusion that generalizes both of the previous diffusions. The general diffusion can model heat flows that behave in essentially arbitrary ways.

The thesis is organized as follows. Table 1.1 summarizes the topics most relevant to the chapters of the thesis. After providing brief background information on common notation and methods

Table 1.1.
An overview of the contributions of the thesis.

	PageRank	Heat kernel	General diffusions
Strong localization	Chapter 3	Chapters 4, 5	Chapter 6
Weak localizaton	Andersen et al. [2006a]	Chapter 7	Chapter 7

used throughout (Chapter 2), we discuss localization in personalized PageRank. In particular we demonstrate that there exist families of graphs for which PageRank exhibits no localized behavior (Section 3.2), then study a class of graphs with properties akin to real-world graphs, and finally prove that PageRank is localized on these networks (Section 3.3). The key technique in our proof involves using the local Gauss-Southwell linear solver method, and studying its rate of convergence.

In Chapter 4, we then propose a new algorithm that modifies Gauss-Southwell to compute the matrix exponential and use this to prove a localization result for the matrix exponential on the same family of graphs. Our adaptation of Gauss Southwell leads us to design two other new algorithms (Chapter 5) which our experimental evaluation shows to be extremely fast compared to the state of the art. In Chapter 6 we present an algorithm for performing a rounded matrix-vector product in a near-optimal way, for which we present a novel greedy approximation algorithm for the Knapsack Problem. Finally, in Chapter 7 we focus on local graph diffusion methods. We generalize a widely-used method for personalized PageRank to compute the heat kernel diffusion, solving an open problem first suggested in [Chung, 2009], we develop a framework for computing a broader class of diffusions, and we prove that both run in constant-time.

All work presented in this thesis has been jointly pursued with my advisor David Gleich. The work in Chapter 3 is joint with Huda Nassar and David Gleich. The work in Chapter 7 is joint with Olivia Simpson and David Gleich.

2. BACKGROUND

We begin with an overview of our notation before describing a few other technical preliminaries common across the chapters of this thesis. Throughout we use $G = (V, E)$ to denote a graph with $n = |V|$ vertices and $m = |E|$ edges. Unless explicitly stated otherwise, all graphs we consider are unweighted, connected, loopless, and undirected. We label a graph's nodes with consecutive integers $1, \cdots, n$ so we can refer to a node by its label j. The degree of a node j is the number of edges incident to that node, and we denote it by $d(j)$ or d_j. If node j is incident to a node i, we denote this by $i \sim j$.

The symbol $\mathbf{A}$ we use to denote a graph's *adjacency matrix*, and it is defined by $\mathbf{A}_{ij} = 1$ if and only if node j has an arc pointing to node i, with a zero otherwise. (Undirected graphs are then symmetric with $\mathbf{A}_{ij} = \mathbf{A}_{ji}$.) For an undirected graph we associate a diagonal *degree matrix*, $\mathbf{D}$, such that $\mathbf{D}_{jj} = d(j)$, and $\mathbf{D}_{ij} = 0$ off the diagonal. The probability transition matrix is then the matrix $\mathbf{P} = \mathbf{A}\mathbf{D}^{-1}$; it has $\mathbf{P}_{ij} = 1/d(j)$ if $j \sim i$, and 0 otherwise.

We follow a standard notation in linear algebra literature and use upper-case bold letters to denote matrices (for example $\mathbf{A}$ above) while lower-case bold letter letters denote vectors. For example, the vector of all 1s we denote by $\mathbf{e}$; the jth standard basis vector we denote by $\mathbf{e}_j$, i.e. the vector of all 0s with a single 1 in the jth entry. We consider all vectors to be column vectors, so that any length k vector $\mathbf{v}$ is $k \times 1$ and its transpose $\mathbf{v}^T$ is $1 \times k$. For a sequence of vectors we use superscripts, i.e. $\mathbf{v}^{(k)}$ to denote the kth vector in the sequence.

Frequently we study processes or properties related to a small set of nodes in a graph in which we are interested, called *seed* nodes and denoted S. Often we represent such a set as a vector normalized to sum to 1, i.e. a *seed vector* $\mathbf{s}$. Typical such seed vectors associated to a set S are the uniform distribution $\frac{1}{|S|}\mathbf{e}_S$ and the degree-weighted distribution $\frac{1}{\mathrm{vol}(S)}\mathbf{D}\mathbf{e}_S$.

For any set of nodes, $S \subseteq V$, we define the *volume* of S to be the sum of the degrees of the nodes in S, denoted $\mathrm{vol}(S) = \sum_{j \in S} d(j)$. Next, we define the *boundary* of $S \subseteq V$ to be the set of edges that have one endpoint inside S and the other endpoint outside S, denoted $\partial(S)$. Finally, the *conductance* of S, denoted $\phi(S)$, is defined by

$$\phi(S) := \frac{|\partial(S)|}{\min\{\mathrm{vol}(S), \mathrm{vol}(V - S)\}}.$$

Conductance can be thought of as measuring the extent to which a set is more connected to itself than the rest of the graph and is one of the most commonly used community detection objectives [Schaeffer, 2007]. In later sections we study the conductance of sets produced by our diffusion algorithms.

2.1 Graph diffusions and matrix functions

Much of this thesis considers properties of or the computation of *graph diffusions*. Here we define graph diffusion, review a few popular diffusions, and study the connection between graph diffusions and matrix functions.

Graph diffusions Conceptually a graph diffusion can be thought of as as follows. Imagine injecting dye into a region of a network and letting the dye flow across edges in the network, gradually decaying each time it crosses an edge. As this spreading process converges (as the dye flow decays to zero), each node will contain an amount of dye. A graph diffusion is a vector whose entries equal the amount of dye contained at each node.

Mathematically, let $\mathbf{s}$ be the dye injector sites, which is called a seed vector for some seed set S from which the diffusion begins. Any stochastic vector $\mathbf{v}$ can be thought of as a distribution of dye across the nodes of a graph; the product $\mathbf{Pv}$ then models a diffusion process in which the dye at a node j spreads uniformly to the neighbors of node j.

We define a diffusion vector $\mathbf{f}$ to be

$$\mathbf{f} = \sum_{k=0}^{\infty} c_k \mathbf{P}^k \mathbf{s} \tag{2.1}$$

where the coefficients $c_0, c_1, \cdots$ can be any fixed sequence of real-valued constants satisfying $c_k \geq 0$ and $\sum_{k=0}^{\infty} c_k = 1$. Note that these conditions on c_k guarantee that the summation in Equation (2.1) is well-defined i.e. that the series converges. Thinking of the terms $\mathbf{P}^k$ as steps of a random walk process, these coefficients c_k can be thought of as encoding the rate at which the diffusion process decays.

The diffusion vector itself can also be thought of as a probability distribution across the nodes of a graph. This is because the diffusion vector $\mathbf{f}$ is stochastic, i.e. $\mathbf{f} \geq 0$ entry-wise and $\mathbf{e}^T\mathbf{f} = 1$. To see why, note that nonnegativity and column-stochasticity of the matrix $\mathbf{P}$ and the vector $\mathbf{s}$ imply that each term $\mathbf{P}^k\mathbf{s}$ is stochastic. Furthermore, because the coefficients c_k sum to 1 we have

$$\mathbf{e}^T \sum_{k=0}^{\infty} c_k \mathbf{P}^k \mathbf{s} \quad = \quad \sum_{k=0} c_k \mathbf{e}^T \mathbf{P}^k \quad = \quad \sum_{k=0}^{\infty} c_k$$

which equals 1, proving that $\mathbf{f}$ is stochastic.

Matrix functions Matrix functions is an area of linear algebra concerned with the behavior of scalar-valued functions (for example, $f(x) = \log(x)$) on square matrices (for an excellent introduction, see [Higham, 2008]). Given a scalar-valued function f, a function of a diagonalizable matrix admits the following definition, paraphrased from Chapter 1 in [Higham, 2008].

Definition 2.1.1 *Let $\boldsymbol{M} \in \mathbb{C}^{n\times n}$ be a diagonalizable matrix with eigendecomposition $\boldsymbol{M} = \boldsymbol{V}\Lambda\,\boldsymbol{V}^{-1}$, and let $f(x)$ be a function defined on the eigenvalues λ_j of $\boldsymbol{M}$. Then we can define*

$$f(\boldsymbol{M}) = \boldsymbol{V} f(\boldsymbol{\Lambda})\,\boldsymbol{V}^{-1}$$

where the quantity $f(\Lambda)$ is defined to be the diagonal matrix with entries $f(\lambda_j)$ on the diagonal.

Note that a function of a diagonal matrix Λ is simply the diagonal matrix with the function applied entry-wise to the diagonal of Λ. Multiple sections of this thesis are concerned with undirected graphs, which have symmetric adjacency matrices; this guarantees that the adjacency matrices $\mathbf{A}$ and random-walk transition matrices $\mathbf{P}$ of undirected graphs are diagonalizable, and so the above definition can be applied.

Certain sections in this thesis are also concerned with directed graphs. Because directed graphs have adjacency and random-walk matrices that are not necessarily symmetric, they are not necessarily diagonalizable; hence we introduce a second definition for functions of matrices for the case of directed graphs. For the specific functions we study, the exponential e^x and the resolvent $(1-\alpha x)^{-1}$ (α is a constant in $(0,1)$), it suffices for our purposes to use the following power series definitions. The matrix exponential for any square matrix can be expressed $e^{\mathbf{M}} = \sum_{k=0}^{\infty} \frac{1}{k!}\mathbf{M}^k$; the series converges for any matrix [Moler and Van Loan, 2003]. The matrix resolvent function can be expressed as the geometric series $(\mathbf{I}-\alpha\mathbf{M})^{-1} = \sum_{k=0}^{\infty} \alpha^k \mathbf{M}^k$ for any matrix $\mathbf{M}$ with norm bounded by 1. In particular, this holds for any random-walk transition matrix $\mathbf{P}$, since they satisfy $\|\mathbf{P}\|_1 \leq 1$.

We remark for the curious reader that Definition 2.1.1 can be extended, albeit with more nuance, to apply to non-diagonalizable matrices (see Chapter 1 in [Higham, 2008]).

2.2 The Gauss Southwell linear solver

One of the primary tools we use in proving our localization results is the Gauss-Southwell linear solver. We also adapt this method in designing some of the algorithms we present in later sections. The Gauss-Southwell (GS) method is an iterative method related to the Gauss-Seidel and coordinate descent methods [Luo and Tseng, 1992]. In solving a linear system $\mathbf{Mx} = \mathbf{b}$ with current solution $\mathbf{x}^{(k)}$ and residual $\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}$, the GS iteration acts via coordinate relaxation on the largest magnitude entry of the residual at each step, whereas the Gauss-Seidel method repeatedly cycles through all elements of the residual. Like Gauss-Seidel, the GS method converges on diagonally dominant matrices, symmetric positive definite matrices, and M-matrices. It is strikingly effective when the underlying system is sparse and the solution vector can be approximated locally. Because of this, the algorithm has been reinvented in the context of computing local graph diffusions like

PageRank [Andersen et al., 2006a, Berkhin, 2007, Jeh and Widom, 2003]. Next we present the basic iteration of GS.

Given a linear system $\mathbf{Mx} = \mathbf{b}$ with initial solution $\mathbf{x}^{(0)} = 0$ and residual $\mathbf{r}^{(0)} = \mathbf{b}$, GS proceeds as follows. To update from step k to step $k+1$, set m_k to be the maximum magnitude entry of $\mathbf{r}^{(k)}$, i.e. $m_k := (\mathbf{r}^{(k)})_{i(k)}$, where $i(k)$ is the index being operated on during step k. Then, update the solution and residual:

$$\begin{aligned} \mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + m_k \cdot \mathbf{e}_{i(k)} && \textit{update the } i(k)\textit{th coordinate only} \\ \mathbf{r}^{(k+1)} &= \mathbf{r}^{(k)} - m_k \cdot \mathbf{M}\mathbf{e}_{i(k)} && \textit{update the residual.} \end{aligned} \tag{2.2}$$

Observe that updating the residual $\mathbf{r}^{(k)}$ in (2.2) involves adding only a scalar multiple of a column of $\mathbf{M}$ to $\mathbf{r}^{(k)}$. The whole step involves updating a single entry of the solution $\mathbf{x}^{(k)}$, and, if $\mathbf{M}$ is sparse, then only a small number of entries of $\mathbf{r}^{(k)}$. When $\mathbf{M}$ is related to a graph matrix (the adjacency matrix, for example) then updating the residual involves accessing the out-links of a single node.

The reason that Gauss-Southwell is called a "coordinate relaxation" method is that it can be derived by *relaxing* or *freeing* the $i(k)$th coordinate to satisfy the linear equations in that coordinate only. For instance, suppose for the sake of simplicity that $\mathbf{M}$ has 1s on its diagonal and let $\mathbf{a}_{i(k)}^T$ be the $i(k)$th row of $\mathbf{M}$. Then at the kth step, we choose $\mathbf{x}^{(k+1)}$ such that $\mathbf{a}_{i_k}^T \mathbf{x}^{(k+1)} = b_{i(k)}$, but we allow only $x_{i(k)}$ to vary – it was the coordinate that was relaxed. Because $\mathbf{M}$ has 1s on its diagonal, we can write this as:

$$x_{i(k)}^{(k+1)} = b_{i(k)} - \sum_{j \neq i_k} (M_{i(k),j}) x_j^{(k)} = r_{i(k)}^{(k)} + x_{i(k)}^{(k)}.$$

This is exactly the same update as in (2.2). It's also the same update as in the Gauss-Seidel method. The difference with Gauss-Seidel, as it is typically explained, is that it does not maintain an explicit residual and it chooses coordinates cyclically.

3. LOCALIZATION IN PAGERANK

Personalized PageRank vectors [Page et al., 1999] are a ubiquitous tool in data analysis of networks in biology [Freschi, 2007, Morrison et al., 2005] and information-relational domains such as recommender systems and databases [Gori and Pucci, 2007, Jain and Pantel, 2010, Nie et al., 2005]. In contrast to the standard PageRank vector, personalized PageRank vectors model a random-walk process on a network that randomly returns to a fixed starting node instead of restarting from a random node in the network as in the traditional PageRank. This process is also called a random-walk with restart. Though this perspective is useful in understanding the utility of PageRank in applications, for our purposes of studying localization properties of PageRank we prefer the perspective of PageRank as a vector times a function of a matrix, specifically the resolvent function $f(x) = (1-\alpha)(1-\alpha \cdot x)^{-1}$, where α is a constant discussed below. (See Figure 3.1 for a visualization of the function.) This chapter of this thesis is then concerned with studying localization in columns of PageRank, i.e. $f(\mathbf{P})\mathbf{e}_c$. We first give a little more background on PageRank before discussing exactly the kind of localization we wish to study.

A personalized PageRank vector is defined from three inputs: the network modeled as a column-stochastic matrix $\mathbf{P}$ characterizing the random-walk process, a parameter $\alpha \in (0, 1)$ that determines the probability $(1-\alpha)$ that the random walk procedure restarts from the seed node, and a seed node s. The personalized PageRank vector $\mathbf{x}$ is then the solution of the linear system:

$$(\mathbf{I} - \alpha\mathbf{P})\mathbf{x} = (1-\alpha)\mathbf{e}_s.$$

From the perspective of matrix functions, we can express PageRank as $\mathbf{x} = f(\mathbf{P})\mathbf{e}_s$, where $f(x)$ is the resolvent function $f(x) = (1-\alpha)(1-\alpha \cdot x)^{-1}$. Recall that the spectrum of $\mathbf{P}$ is contained in the interval $[-1, 1]$, with 1 as its largest magnitude eigenvalue. To understand how $f(x)$ acts on the spectrum of $\mathbf{P}$, in Figure 3.1 we display the resolvent on the interval $[-1, 1]$ for four values of α. The plot shows that as α approaches 1, the resolvent more heavily weights values of x near 1; since the input matrix $\mathbf{P}$ has 1 as its largest magnitude eigenvalue, this means the resolvent applied to $\mathbf{P}$ dampens the non-dominant eigenvalues of $\mathbf{P}$.

When the network is strongly connected, the solution $\mathbf{x}$ is non-zero for all nodes. This is because there is a non-zero probability of walking from the seed to any other node in a strongly connected network. Nevertheless, the solution $\mathbf{x}$ displays a behavior called *localization*. We can attain accurate localized PageRank solutions by truncating small elements of $\mathbf{x}$ to zero. Put another way, there is a sparse vector $\hat{\mathbf{x}}$ that approximates $\mathbf{x}$ to an accuracy of ε. This behavior is desirable for applications

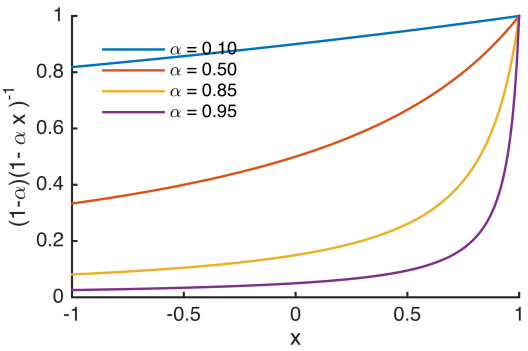

Figure 3.1. The resolvent function.

of seeded PageRank because they typically seek to "highlight" a small region related to the seed node s inside a large graph. Figure 3.2 illustrates an example of such localization behavior on a version of the DBLP network obtained from the SNAP repository [Yang and Leskovec, 2012]. The left plot displays the values in a seeded PageRank vector computed with $\alpha = 0.6$; it shows that only a very small number of nodes have large PageRank value. The right plot shows the 1-norm error of an approximation to the true PageRank vector $\mathbf{x}$ that uses only the x largest entries in $\mathbf{x}$.

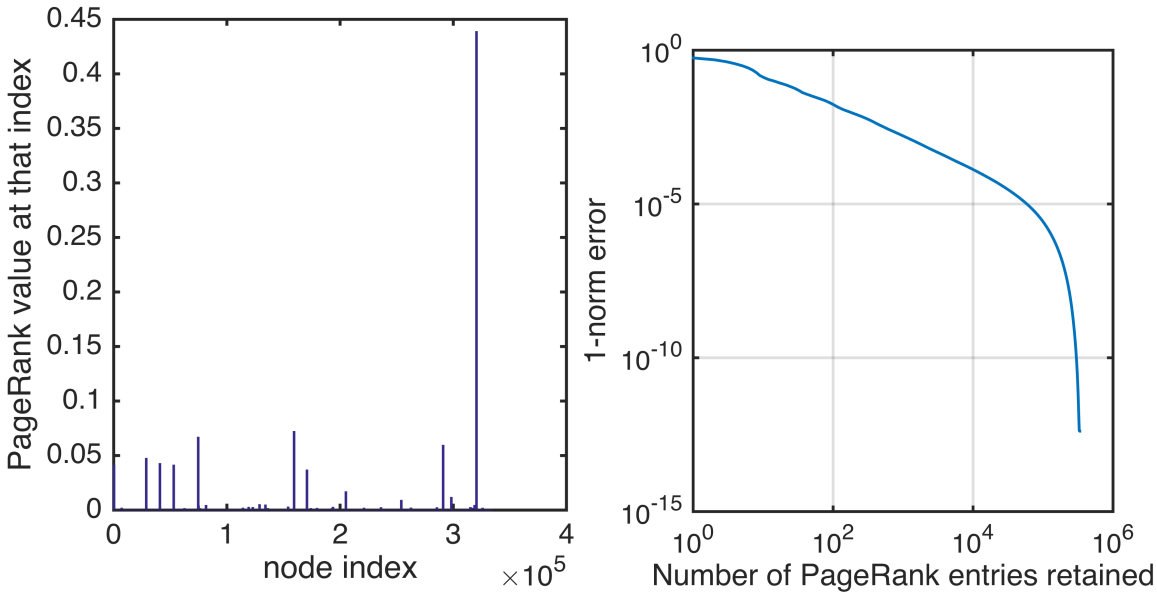

Figure 3.2. Localization in seeded PageRank on the DBLP graph.

The essential question we study in this paper is: for a given accuracy ε how sparse can we make the approximation $\hat{\mathbf{x}}$? To be precise, we consider a notion of strong localization, $\|\hat{\mathbf{x}} - \mathbf{x}\|_1 \leq \varepsilon$, and

we focus on the behavior of $\min \text{nonzeros}(\hat{\mathbf{x}})$. Note that $\hat{\mathbf{x}}$ depends on α, the particular random-walk on the graph $\mathbf{P}$, and the seed node s from which the PageRank diffusion begins. We only consider stochastic matrices $\mathbf{P}$ that arise from random-walks on strongly-connected graphs. So a more precise statement of our goal is:

$$\max_{\mathbf{P}} \max_{s} \min_{\hat{\mathbf{x}}} \text{nonzeros}(\hat{\mathbf{x}}) \text{ where } \|\hat{\mathbf{x}} - \mathbf{x}(\alpha, \mathbf{P}, s)\|_1 \leq \varepsilon,$$

and where $\mathbf{x}(\alpha, \mathbf{P}, s)$ is the personalized PageRank vector $(1-\alpha)(\mathbf{I} - \alpha\mathbf{P})^{-1}\mathbf{e}_s$. The goal is to establish bounds on this number of nonzeros that are sublinear in n, the number of nodes of the graph, because that implies localized solutions to PageRank. We remark that the localization results in this chapter apply to both directed and undirected graphs, but our de-localization results apply to a specific family of undirected graphs.

3.1 Related work

Related work on weak localization There is another notion of localization that appears in uses of PageRank for partitioning undirected graphs:

$$\left\|\mathbf{D}^{-1}(\hat{\mathbf{x}} - \mathbf{x})\right\|_\infty$$

If this notion is used for a localized Cheeger inequality [Andersen et al., 2006a, Chung, 2007a], then we need the additional property that $0 \leq \hat{\mathbf{x}} \leq \mathbf{x}$ element-wise. When restated as a localization result, the famous Andersen-Chung-Lang PageRank partitioning result [Andersen et al., 2006a] includes a proof that:

$$\max_{\mathbf{P}} \max_{s} \min_{\hat{\mathbf{x}}} \text{nonzeros}(\hat{\mathbf{x}}) \leq \tfrac{1}{1-\alpha}\tfrac{1}{\varepsilon}, \text{ where } \left\|\mathbf{D}^{-1}(\hat{\mathbf{x}} - \mathbf{x}(\alpha, \mathbf{P}, s))\right\|_\infty \leq \varepsilon.$$

This establishes that *any* uniform random walk on a graph satisfies a weak-localization property. The paper also gives a fast algorithm to find these weakly local solutions. We explore this notion of localization in the heat kernel and other diffusions in Chapter 7.

Related work on functions of matrices and diffusions Localization in diffusions is broadly related to localization in functions of matrices [Benzi et al., 2013]. The results in that literature tend to focus on the case of banded matrices (e.g. [Benzi and Razouk, 2007]), although there are also discussions of more general results in terms of graphs arising from sparse matrices [Benzi et al., 2013]. These same types of decay bounds can apply to a variety of graph diffusion models that involve a stochastic matrix [Baeza-Yates et al., 2006, Huberman et al., 1998], and recent work shows that they may even extend beyond this regime [Ghosh et al., 2014]. In the context of the decay of functions of matrices, we advance the literature by proving a localization bound for a particular resolvent function of a matrix that applies to graphs with growing maximum degree.

3.2 Negative results for strong localization

Much of the work in this thesis focuses on proving that certain graphs have localized properties, i.e. that a given diffusion on a graph will be highly concentrated in a very small region. For such results to be interesting, we want to know that this is not always the case – we want to know that there are graphs in which no such localization occurs (otherwise, if localization occurs in all graphs, the results proved later in this manuscript would not be at all surprising). In this section we construct families of graphs for which many diffusions and matrix functions do not exhibit local behavior. In particular, we show in Proposition 3.2.1 that for all complete bipartite graphs, many matrix function are de-localized.

The results presented in this section are specific (and more powerful) instances of the following broader facts. Given an undirected graph, its random walk transition $\mathbf{P} = \mathbf{A}\mathbf{D}^{-1}$ is necessarily diagonalizable, because it is a similarity transformation of a symmetric matrix: $\mathbf{D}^{-1/2}\mathbf{P}\mathbf{D}^{1/2} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ (where $\mathbf{A}$ is symmetric because we have assumed the graph is undirected). Consequently we can use the following simple result from the theory of functions of matrices ([Higham, 2008], Chapter 1.2): for any function $f(x)$ defined on the spectrum of $\mathbf{P}$, we can express $f(\mathbf{P}) = p(\mathbf{P})$ for any polynomial $p(x)$ that interpolates the function $f(x)$ on the spectrum of $\mathbf{P}$. If $\mathbf{P}$ has r distinct eigenvalues (i.e. $\mathbf{P}$ has a minimal polynomial of degree r), then this implies that $f(\mathbf{P})$ can be expressed using an interpolating polynomial on r values, and so $p(x)$ can be a polynomial of degree $r-1$.

Using the above facts, any matrix function $f(\mathbf{P})$ (that is defined on the spectrum of $\mathbf{P}$) can be expressed in the form

$$f(\mathbf{P}) = c_0\mathbf{I} + c_1\mathbf{P} + \cdots + c_{r-1}\mathbf{P}^{r-1}$$

for appropriate coefficients c_j. These standard facts from the theory of functions of matrices will combine with assumptions on graph structure to produce our de-localization results.

We remark that the results discussed in this section are the results of collaboration with Huda Nassar and David F. Gleich, and are generalizations of results first published with those co-authors in [Nassar et al., 2015], where we proved the smaller result that personalized PageRank vectors are de-localized on star graphs. Here we consider the entire family of complete bipartite graphs (which contains the set of star graphs), and we address a broader set of functions than just the PageRank function.

3.2.1 Complete bipartite graphs

The random walk transition matrix $\mathbf{P}$ of any complete bipartite graph has eigenvalues $-1, 0$, and 1 ([Chung, 1997], Chapter 1.2). As noted above, $\mathbf{P}$ is diagonalizable, so we can determine $f(\mathbf{P})$ by the action of $f(x)$ on the spectrum of $\mathbf{P}$. Thus, for any function $f(x)$ that is defined on the values $\{-1, 0, 1\}$, and any polynomial $p(x)$ that satisfies

$$p(-1) = f(-1) \quad p(0) = f(0) \quad p(1) = f(1) \tag{3.1}$$

we will also have $p(\mathbf{P}) = f(\mathbf{P})$. Below we give the structure of the interpolating polynomial for a general function before showing how such polynomials can be used to prove that these functions are not local on complete bipartite graphs.

The intuition of our results is that, because complete bipartite graphs have 3 eigenvalues, any function of the graph can be expressed using an interpolating polynomial of degree 2; the matrices $\mathbf{P}$ and $\mathbf{P}^2$ then determine the structure of any function $f(\mathbf{P})$, and because of the uniform nature of complete bipartite graphs, the expressions $\mathbf{P}$ and $\mathbf{P}^2$ (and, hence, $f(\mathbf{P})$) are roughly the sum of uniform distributions and so cannot be approximated using only small number of entries their non-zero.

Interpolating polynomials Given any function $f(x)$ defined on the values $\{-1, 0, 1\}$, the degree 2 interpolating polynomial $p(x)$ for this function, defined by Equation (3.1) has coefficients $p(x) = c_0 + c_1 x + c_2 x^2$ given by

$$c_0 = f(0) \tag{3.2}$$

$$c_1 = \tfrac{1}{2}(f(1) - f(-1)) \tag{3.3}$$

$$c_2 = \tfrac{1}{2}(f(1) + f(-1) - 2f(0)). \tag{3.4}$$

The value $c_0 = f(0)$ follows from plugging 0 into $p(x)$ and noting $f(0) = p(0) = c_0 + 0 + 0$. The other values follow from similar straight-forward algebra. Thus, for a general function $f(x)$ defined on the spectrum of a complete bipartite graph, we can study the localization of

$$f(\mathbf{P}) = f(0) \cdot \mathbf{I} + \left(\tfrac{1}{2}(f(1) - f(-1))\right) \cdot \mathbf{P} + \left(\tfrac{1}{2}(f(1) + f(-1) - 2f(0))\right) \cdot \mathbf{P}^2, \tag{3.5}$$

by looking at the structure of the matrices $\mathbf{P}$ and $\mathbf{P}^2$.

Graph structure Here we will consider a complete bipartite graph with n nodes divided into a partition of k nodes and $n - k$ nodes. In particular we want to understand the structure of the random walk transition matrix, $\mathbf{P}$, as well as $\mathbf{P}^2$ for such graphs. The nodes can be ordered so that

the adjacency matrix has the block form $\mathbf{A} = \begin{bmatrix} 0 & \mathbf{E}^T \\ \mathbf{E} & 0 \end{bmatrix}$, where $\mathbf{E}$ is the $(n-k) \times k$ matrix of all 1s. The degree matrix is then $\mathbf{D} = \begin{bmatrix} (n-k) \cdot \mathbf{I}_{n-k} & 0 \\ 0 & k \cdot \mathbf{I}_k \end{bmatrix}$. Letting $\begin{bmatrix} \mathbf{I}_k \\ 0 \end{bmatrix}$ be a $n \times k$ matrix with a $k \times k$ identity block in the upper-most block, we can express the relationships

$$\mathbf{P} = \mathbf{A}\mathbf{D}^{-1} = \tfrac{1}{k} \begin{bmatrix} \mathbf{I}_k \\ 0 \end{bmatrix} \begin{bmatrix} 0 & \mathbf{E}^T \end{bmatrix} + \tfrac{1}{n-k} \begin{bmatrix} 0 \\ \mathbf{E} \end{bmatrix} \begin{bmatrix} \mathbf{I}_k & 0 \end{bmatrix} \tag{3.6}$$

$$= \tfrac{1}{k} \begin{bmatrix} 0 & \mathbf{E}^T \\ 0 & 0 \end{bmatrix} + \tfrac{1}{n-k} \begin{bmatrix} 0 & 0 \\ \mathbf{E} & 0 \end{bmatrix} \tag{3.7}$$

and

$$\mathbf{P}^2 = \tfrac{1}{n-k}\tfrac{1}{k} \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{E}\mathbf{E}^T \end{bmatrix} + \tfrac{1}{n-k}\tfrac{1}{k} \begin{bmatrix} \mathbf{I}_k \\ 0 \end{bmatrix} \mathbf{E}^T\mathbf{E} \begin{bmatrix} \mathbf{I}_k & 0 \end{bmatrix} \tag{3.8}$$

$$= \tfrac{1}{n-k} \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{J}_{n-k} \end{bmatrix} + \tfrac{1}{k} \begin{bmatrix} \mathbf{J}_k & 0 \\ 0 & 0 \end{bmatrix} \tag{3.9}$$

where $\mathbf{J}_i$ is a square block of 1s of dimension $i \times i$. (We use $\mathbf{J}_i$ specifically for square matrices of all 1s, and $\mathbf{E}$ for matrices of all 1s that are possibly not square.) Equation (3.9) uses the relationships $\mathbf{E}\mathbf{E}^T = k \cdot \mathbf{J}_{n-k}$ and $\mathbf{E}^T\mathbf{E} = (n-k) \cdot \mathbf{J}_k$, which follow from the fact that the matrix $\mathbf{E}$ is all 1s and dimension $(n-k) \times k$.

The expressions for $\mathbf{P}$ and $\mathbf{P}^2$ in Equations (3.7) and (3.9) enable us to show that columns of matrix functions $f(\mathbf{P})\mathbf{e}_j$ are de-localized, except for extreme cases in which we discuss below. To show this, we will simply substitute the above expressions for $\mathbf{P}^t$ in Equations (3.7) and (3.9) into Equation (3.5) and perform a little algebra. First consider any node j in the partition containing k nodes. From the structure of $\mathbf{P}$ described above, we get that $\mathbf{P}\mathbf{e}_j = \frac{1}{n-k} \begin{bmatrix} 0 \\ \mathbf{e} \end{bmatrix}_{n-k}$, where $\begin{bmatrix} 0 \\ \mathbf{e} \end{bmatrix}_{n-k}$ is a vector with k 0s and $n-k$ 1s in the lower block. Similarly we obtain that $\mathbf{P}^2\mathbf{e}_j = \frac{1}{k} \begin{bmatrix} \mathbf{e} \\ 0 \end{bmatrix}_k$, where $\begin{bmatrix} \mathbf{e} \\ 0 \end{bmatrix}_k$ is a vector with k 1s in the upper block, followed by $(n-k)$ zero entries.

Now we use this information to compute the magnitude of the individual entries of the vectors $f(\mathbf{P})\mathbf{e}_j$. Substituting into Equation (3.5) the structure of $\mathbf{P}\mathbf{e}_j$ and $\mathbf{P}^2\mathbf{e}_j$ that we just computed, we can see that $f(\mathbf{P})\mathbf{e}_j$ has k entries of magnitude exactly $\frac{1}{k}c_2$, and $n-k-1$ entries of magnitude exactly $\frac{1}{n-k}c_1$, and finally entry j itself has magnitude $f(0) + \frac{1}{n-k}c_1$.

The purpose of this nitty gritty arithmetic is that it allows us to see that a vector $f(\mathbf{P})\mathbf{e}_j$ has too many non-trivial entries, and so cannot be approximated to an accuracy of ε with fewer than $O(n)$

of its non-zero entries, unless the function $f(x)$ is such that at least one of the coefficients $\frac{1}{n-k}c_1$ or $\frac{1}{k}c_2$ is very small relative to ε (where the coefficients c_t are as defined above in Equation (3.2)). We make this notion rigorous in the following proposition, which the above discussion has proved:

Proposition 3.2.1 *Let $\boldsymbol{P}$ be the random walk transition matrix of a complete bipartite graph on n nodes, and let the partition sizes be k and $n-k$. Let $f(x)$ be any function defined on the spectrum of a complete bipartite graph, $\{-1, 0, 1\}$. Then columns of $f(\boldsymbol{P})$ can be expressed as follows. For any node j in the partition of size k, we have, in the notation of Section 3.2.1,*

$$f(\boldsymbol{P})\boldsymbol{e}_j = c_0 \cdot \boldsymbol{e}_j + \tfrac{1}{n-k}c_1 \cdot \begin{bmatrix} 0 \\ \boldsymbol{e} \end{bmatrix}_{n-k} + \tfrac{1}{k}c_2 \cdot \begin{bmatrix} \boldsymbol{e} \\ 0 \end{bmatrix}_{k},$$

where the coefficients c_k are defined by

$$c_0 = f(0); \quad c_1 = \tfrac{1}{2}(f(1) - f(-1)) \quad c_2 = \tfrac{1}{2}(f(1) + f(-1) - 2f(0)).$$

The column $f(\mathbf{P})\mathbf{e}_j$ consists of one entry of magnitude $c_0 + c_2/k$, $(k-1)$ entries of magnitude c_2/k, and $(n-k)$ entries of magnitude $c_1/(n-k)$. Thus, every column of $f(\mathbf{P})$ consists of a diagonal entry $f(\mathbf{P})_{jj}$ and one sub-vector for each of the two graph partition, with each sub-vector having entries of uniform value. It is possible that the function f behaves such that a vector $f(\mathbf{P})\mathbf{e}_j$ is localized. For example, if k is very small and $c_2 = 1 - \varepsilon$, then the entries outside of the partition of size k can all be rounded to zero and the resulting approximation would still have error bounded above by ε. A precise statement of the number of nonzeros necessary to obtain a specified accuracy would require information about the coefficients c_j.

We remark that the above result applies to nodes in either partition of the complete bipartite graph, since we make no assumption the size of partition with k nodes, i.e. is it the larger or smaller side. Hence, by symmetry, a statement about the de-localization of columns of $f(\mathbf{P})$ corresponding to nodes on the partition of size k will really apply to all columns of $f(\mathbf{P})$.

PageRank on a star graph Next we demonstrate Proposition 3.2.1 in the specific case of PageRank on a star graph, to illustrate more concretely the utility of our result. A star graph has one partition with one node (the center node) and one partition with $n-1$ nodes (all nodes other than the center). We want to explicitly demonstrate de-localization of the PageRank function $f(x) = (1-\alpha)(1-\alpha \cdot x)^{-1}$ on the star graph.

Using the formulae given in the above proposition we can express a column $\mathbf{x} = f(\mathbf{P})\mathbf{e}_j$ explicitly. First assume that the node j is the center node, $j = 1$ (so that in the statement of Proposition 3.2.1 we have $k = 1$, the small side of the graph). Then plugging values into the formulae and simplifying yields $\mathbf{x} = \frac{1}{1+\alpha}\mathbf{e}_1 + \frac{1}{n-1}\frac{\alpha}{1+\alpha}\begin{bmatrix} 0 \\ \mathbf{e} \end{bmatrix}_{n-1}$. Thus, the PageRank vector $\mathbf{x}$ seeded on the center node has

value $1/(1+\alpha)$ for the center node and $\alpha/((1+\alpha)(n-1))$ for all leaf nodes. Suppose an approximation $\hat{\mathbf{x}}$ of $\mathbf{x}$ has M of these leaf-node entries set to 0. Then the 1-norm error $\|\mathbf{x}-\hat{\mathbf{x}}\|_1$ would be at least $M\alpha/((1+\alpha)(n-1))$. Attaining a 1-norm accuracy of ε requires $M\alpha/((1+\alpha)(n-1)) < \varepsilon$, and so the minimum number of entries of the approximate PageRank vector required to be non-zero $(n-M)$ is then lower-bounded by $n(1-c)+c$, where $c = \varepsilon(1+\alpha)/\alpha$. Note that this requires $c \in (0,1)$, which holds if $\varepsilon < \alpha/2$. Thus, the number of nonzeros in the approximation must be linear in n.

This example shows how simple use of the formulae in Proposition 3.2.1 enabled a quick and more intuitive proof of our result in [Nassar et al., 2015], which relied more heavily on computations instead of appealing to the spectrum of the graph.

Future Work

In this section we discussed the de-localization of functions on highly structured graphs (namely, complete bipartite graphs). Interestingly, for these graphs we also saw that the functions of their random walk transition matrices turned out to have off-diagonal blocks with low rank (rank 1, in fact, because the matrix $\mathbf{J}_k$ is simply $\mathbf{e}\mathbf{e}^T$). If this phenomenon of low-rank off-diagonal block structure occurs on broader types of graphs (i.e. other than complete bipartite), the study of de-localization of functions on graphs could potentially lead to useful tools in graph compression.

Broader graph classes The results above rely on a few key properties of the underlying problem. In particular, the ability to express $f(\mathbf{P})$ with an interpolating polynomial of small degree plays an important role. This in turn relies on complete bipartite graphs having such a structured spectrum (i.e. $\lambda_j = -1, 0, 1$). Additionally, the matrices $\mathbf{P}$ and $\mathbf{P}^2$ have exactly known nonzero patterns, enabling exact computation of the size of each nonzero in the personalized PageRank vector.

To adapt our proof to more complicated kinds of graphs, we need to be able to obtain comparable properties. One way to proceed is to assume that our graph is a low-rank update of a complete-bipartite graph. In particular, if we assume that our graph is a rank-2 update of a complete bipartite graph, then we can simply use our above theory for complete bipartite graphs to obtain $(1-\alpha)(\mathbf{I}-\alpha\mathbf{P})^{-1}$, and then use the Sherman-Morrison-Woodbury Identity to write $(\mathbf{I}-\alpha\mathbf{P}_{new})^{-1}$ in terms of $(\mathbf{I}-\alpha\mathbf{P})^{-1}$ and other easily computable terms. This strategy could generalize our de-localization result from PageRank on complete bipartite graphs to PageRank on graphs that are within one edge of being complete-bipartite. This is because a graph that results from adding or deleting an edge from a complete bipartite graph can be expressed as follows:

$$\mathbf{P}_{\text{new}} = \mathbf{P} + \mathbf{c}_{j_1}\mathbf{e}_{j_1}^T + \mathbf{c}_{j_2}\mathbf{e}_{j_2}^T$$

where the vectors $\mathbf{c}_j$ add a single edge from node j_1 to node j_2 and update the other entries in those columns so they are properly scaled. More specifically, to update a single node's column in $\mathbf{P}$, add $(\mathbf{c}_{\text{new}} - \mathbf{c}_{\text{old}})\mathbf{e}_j^T$ to $\mathbf{P}$. The Sherman-Morrison-Woodbury identity then would enable us to correct the PageRank matrix $(\mathbf{I} - \alpha\mathbf{P})^{-1}$ with only a rank 2 update. Alternatively, we can think of this as causing a correction to the PageRank vectors themselves; the correction consists of adding a de-localized vector to the already de-localized personalized PageRank vectors, so the result will still be de-localized. This approach works in particular for the PageRank matrix because it is based on an inverse. Other functions of matrices will require other updates.

3.3 Localization in Personalized PageRank

The example in Section 3.2 demonstrates that there exist matrix functions with columns that are *non-local* in the strong sense (and, in particular, that there exist seeded PageRank vectors that are not localized). Here we show that graphs with a particular type of skewed degree sequence and a growing, but sublinear, maximum degree have seeded PageRank vectors that are always localized, and we give an upper-bound on $f(\varepsilon)$ for this class of graph. The skewed degree sequence is related to the power-law degree distribution commonly observed in many types of real-world networks. With slight improvements to our results, it is possible they could be strengthened to apply graphs with a power-law degree distribution. We first state and discuss our main result in this section, then give the details of the proof.

Theorem 3.3.1 *Let $\boldsymbol{P}$ be a uniform random walk transition matrix of a graph on n nodes with maximum degree d and minimum degree δ. Additionally, suppose that the kth largest degree, $d(k)$, satisfies $d(k) \leq \max\{dk^{-p},\ \delta\}$. The Gauss-Southwell coordinate relaxation method applied to the seeded PageRank problem $(\boldsymbol{I}-\alpha\boldsymbol{P})\boldsymbol{x} = (1-\alpha)\boldsymbol{e}_s$ produces an approximation $\boldsymbol{x}_\varepsilon$ satisfying $\|\boldsymbol{x}-\boldsymbol{x}_\varepsilon\|_1 < \varepsilon$ having at most N non-zeros in the solution, where N satisfies*

$$N = \min\left\{\ n\ ,\ \tfrac{1}{\delta}C_p\left(\tfrac{1}{\varepsilon}\right)^{\frac{\delta}{1-\alpha}}\right\}, \tag{3.10}$$

and where we define C_p to be

$$\begin{aligned} C_p &:= d(1+\log d) && \text{if } p = 1 \\ &:= d\left(1 + \tfrac{1}{1-p}\left(d^{\frac{1}{p}-1} - 1\right)\right) && \text{otherwise.} \end{aligned}$$

Note that the upper bound $N = n$ is trivial as a vector cannot have more non-zeros than entries. Thus, d, δ, p, and n must satisfy certain conditions to ensure that inequality (3.10) is not trivial. In particular, for values of $p < 1$, it is necessary that $d = o(n^p)$ for inequality (3.10) to imply that

$N = o(n)$. For $p > 1$, the bound guarantees sublinear growth of N as long as $d = o(n)$. Additionally, the minimum degree δ must be bounded by $O(\log \log n)$. Thus we arrive at:

Corollary 3.3.1 *Let G be a class of graphs with degree sequences obeying the conditions of Theorem 3.3.1 with constant δ and $d = o(n^{\min(p,1)})$. Then $f(\varepsilon) = o(n)$, and seeded PageRank vectors are localized.*

We also note that the theorem implies localized seeded PageRank vectors for any graph with a maximum degree $d = O(\log \log n)$. This improves slightly on recent work that has demonstrated localization in functions on matrices with constant bandwidth.

3.3.1 Our class of skewed degree sequences

We wish to make a few remarks about the class of skewed degree sequences where our results apply. Perhaps the most well-known is the power-law degree distribution where the probability that a node has degree k is proportional to $k^{-\gamma}$. These power-laws can be related to our skewed sequences with $p = 1/(\gamma - 1)$ and $d = O(n^p)$ [Avrachenkov et al., 2012]. This setting renders our bound trivial with n nonzeros. Nevertheless, there is evidence that some real-world networks exhibit our type of skewed degrees [Faloutsos et al., 1999b] where the bound is asymptotically non-trivial. Figure 3.3 illustrates the kind of skewed degree sequence that we consider. The figure shows the degree sequence for a version of the YouTube network from the SNAP repository [Yang and Leskovec, 2012]; the kth largest degree, $d(k)$, roughly satisfies $d(k) \leq d(1) \cdot k^{-p}$ with an exponent of $p \approx 0.71$.

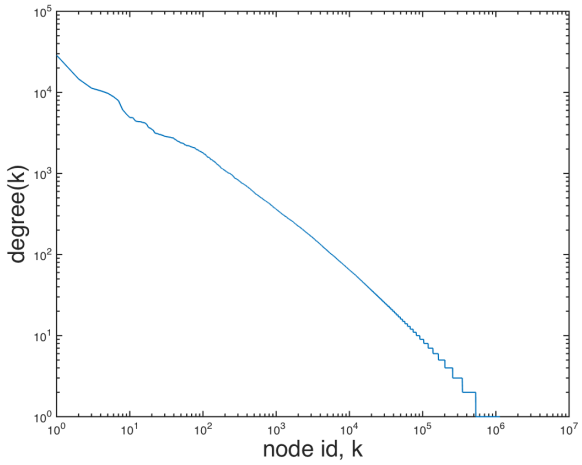

Figure 3.3. Skewed degree sequence in the Youtube network.

3.3.2 Deriving the localization bound

Getting back to the proof, our goal is an ε-approximation, $\mathbf{x}_\varepsilon$, to the equation $(\mathbf{I} - \alpha\mathbf{P})\,\mathbf{x} = (1-\alpha)\mathbf{e}_s$ for a seed s. Given an approximation, $\hat{\mathbf{x}}$, we can express the error in terms of the residual vector $\mathbf{r} = (1-\alpha)\mathbf{e}_s - (\mathbf{I} - \alpha\mathbf{P})\hat{\mathbf{x}}$ as follows:

$$\mathbf{x} - \hat{\mathbf{x}} \quad = \quad (\mathbf{I} - \alpha\mathbf{P})^{-1}\,\mathbf{r}. \tag{3.11}$$

Using this relationship, we can bound our approximation's 1-norm accuracy, $\|\mathbf{x} - \hat{\mathbf{x}}\|_1$, with the quantity $\frac{1}{1-\alpha}\|\mathbf{r}\|_1$. This is because the column-stochasticity of $\mathbf{P}$ implies that $\|(\mathbf{I} - \alpha\mathbf{P})^{-1}\|_1 = \frac{1}{1-\alpha}$. Guaranteeing a 1-norm error $\|\mathbf{x} - \hat{\mathbf{x}}\|_1 < \varepsilon$ is then a matter of ensuring that $\|\mathbf{r}\|_1 < (1-\alpha)\varepsilon$ holds. To bound the residual norm, we will look more closely at using the Gauss-Southwell method for producing the approximation.

The importance of Gauss-Southwell to our main result is as follows: we will use the Gauss-Southwell iteration to compute an approximation $\mathbf{x}_\varepsilon$, and bound the number of iterations necessary to obtain the desired accuracy ε. In doing so, we will bound the number of non-zero entries in $\mathbf{x}_\varepsilon$, justifying our main result.

The Gauss-Southwell iteration on PageRank The Gauss-Southwell algorithm is a sparse linear solver which we discussed in Section 2.2. We review it here in the specific context of PageRank.

Briefly, when solving a linear system, the Gauss-Southwell method proceeds by updating the entry of the approximate solution that corresponds to the largest magnitude entry of the residual, $\mathbf{r}$. The algorithm begins by setting the initial solution $\mathbf{x}^{(0)} = 0$ and $\mathbf{r}^{(0)} = (1-\alpha)\mathbf{e}_s$. In step k, let $j = j(k)$ be the entry of $\mathbf{r}^{(k)}$ with the largest magnitude, and let $m = |\mathbf{r}_j^{(k)}|$. We update the solution $\mathbf{x}^{(k)}$ and residual as follows:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + m\mathbf{e}_j \tag{3.12}$$

$$\mathbf{r}^{(k+1)} = \mathbf{e}_s - (\mathbf{I} - \alpha\mathbf{P})\mathbf{x}^{(k+1)}, \tag{3.13}$$

and the residual update can be expanded to $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - m\mathbf{e}_j + m\alpha\mathbf{P}\mathbf{e}_j$. Since each update to the solution $\mathbf{x}^{(k)}$ alters exactly one entry of the vector, the index k is an upper-bound on the number of non-zeros in the solution.

This application of Gauss-Southwell to seeded PageRank-style problems has appeared numerous times in recent literature [Berkhin, 2007, Bonchi et al., 2012, Jeh and Widom, 2003, McSherry, 2005]. In at least one instance ([Bonchi et al., 2012], Section 5.2) the authors showed that the residual and solution vector stay nonnegative throughout this process, assuming the seed vector is nonnegative (which, in our context, it is). So the 1-norm of the residual can be expressed as $\|\mathbf{r}^{(k+1)}\|_1 = \mathbf{e}^T\mathbf{r}^{(k+1)}$, where $\mathbf{e}$ is the vector of all ones. Expanding the residual in terms of the iterative update presented

above, we can write the residual norm as $\mathbf{e}^T\left(\mathbf{r}^{(k)} - m\mathbf{e}_j + m\alpha\mathbf{P}\mathbf{e}_j\right)$. Then, denoting $\|\mathbf{r}^{(k)}\|_1$ by r_k, yields the recurrence $r_{k+1} = r_k - m(1-\alpha)$.

Next observe that since m is the largest magnitude entry in $\mathbf{r}$, it is larger than the average value of $\mathbf{r}$. Let $Z(k)$ denote the number of nonzero entries in $\mathbf{r}^{(k)}$; then the average value can be expressed as $r_k/Z(k)$. Hence, we have $m \geq r_k/Z(k)$, and so we can bound $r_k - m(1-\alpha)$ above by $r_k - r_k(1-\alpha)/Z(k)$. Thus, $r_{k+1} \leq r_k\left(1-(1-\alpha)/Z(k)\right)$, and we can recur to find:

$$r_{k+1} \leq r_0 \prod_{t=0}^{k}\left(1 - \tfrac{1-\alpha}{Z(t)}\right), \tag{3.14}$$

where $r_0 = (1-\alpha)$ because $\mathbf{r}_0 = (1-\alpha)\mathbf{e}_s$. Then, using the fact that $\log(1-x) \leq -x$ for $x < 1$, we note:

$$r_{k+1} \leq (1-\alpha)\prod_{t=0}^{k}\left(1 - \tfrac{1-\alpha}{Z(t)}\right) \leq (1-\alpha)\exp\left(-(1-\alpha)\sum_{t=0}^{k}\tfrac{1}{Z(t)}\right). \tag{3.15}$$

To progress from here we need some control over the quantity $Z(t)$ and this is where our skewed degree sequence enters the proof.

3.3.3 Using the degree sequence

If we assume that the graph has a skewed degree distribution much like a power-law degree distribution, then we can bound $d(k) \leq d \cdot k^{-p}$ for a constant p in $(0.5, 1]$. Let δ denote the minimum degree of the graph and note that, for sparse networks in which $|E| = O(n)$, δ is a small constant (which, realistically, is $\delta = 1$ in real-world networks). With these facts in place, we will bound $Z(t)$ in terms of d, δ, and p.

Lemma 3.3.1 *Define*

$$C_p = \begin{cases} d(1+\log d) & \text{if } p = 1 \\ d\left(1 + \frac{1}{(1-p)}\left(d^{\frac{1}{p}-1} - 1\right)\right) & \text{if } p \in (0,1)\ . \end{cases} \tag{3.16}$$

Then in the notation described above we have

$$Z(t) \leq C_p + \delta t. \tag{3.17}$$

We presented a similar analysis in [Gleich, 2015b], but the current presentation improves our original bound on C_p. We later prove this bound below, but first we use the bound (3.17) on $Z(t)$ to control the bound on r_k. First we establish the inequality

$$\sum_{t=b}^{k}\tfrac{1}{Z(t)} \geq \tfrac{1}{\delta}\log\left((\delta(k+1)+C_p)/(\delta\cdot b + C_p)\right). \tag{3.18}$$

To see this, first note that (3.17) implies $1/Z(t) \geq (C_p + \delta \cdot t)$, and so for constants m and b we have $\sum_{t=b}^{m} 1/Z(t) \geq \sum_{t=b}^{m} 1/(C_p + \delta \cdot t)$. A left-hand rule integral approximation yields Inequality (3.18). We are interested in particular in the value $b = 0$: $\sum_{t=0}^{k} \frac{1}{Z(t)} \geq \frac{1}{\delta} \log\left((\delta(k+1) + C_p)/C_p\right)$.

Using Inequality (3.18) with $b = 0$. and plugging into (3.15), we can bound

$$r_{k+1} \leq (1-\alpha) \exp\left(-\tfrac{(1-\alpha)}{\delta} \log\left(\tfrac{(\delta(k+1)+C_p)}{C_p}\right)\right),$$

which simplifies to $r_{k+1} \leq (1-\alpha)\left((\delta(k+1) + C_p)/C_p\right)^{(\alpha-1)/\delta}$. Finally, to guarantee $r_k < \varepsilon(1-\alpha)$, it suffices to choose k so that $\left((\delta k + C_p)/C_p\right)^{(\alpha-1)/\delta} \leq \varepsilon$. This holds if and only if $(\delta k + C_p) \geq C_p \left(1/\varepsilon\right)^{\delta/(\alpha-1)}$ holds, which is guaranteed by $k \geq \frac{1}{\delta} C_p \left(1/\varepsilon\right)^{\delta/(1-\alpha)}$. Thus, $k = \frac{1}{\delta} C_p \left(1/\varepsilon\right)^{\delta/(1-\alpha)}$ steps will produce an ε-approximation. Each step introduces at most one non-zero, which implies that if $k < n$, then there is an approximation $\mathbf{x}_\varepsilon$ with $N = k < n$ non-zeros. If $k \geq n$, then this analysis produces the trivial bound $N = n$. To prove these results we assumed Inequality (3.17) without proof. Next we prove this inequality.

Proving the degree sequence bound. Here we prove Lemma 3.3.1. First, observe that the number of nonzeros in the residual after t steps is bounded above by the sum of the largest t degrees, $Z(t) \leq \sum_{k=1}^{t} d(k)$. When we substitute the decay bound $d(k) \leq dk^{-p}$ into this expression, $d(k)$ is only a positive integer when $k \leq (d/\delta)^{1/p}$. Hence, we split the summation $Z(t) \leq \sum_{k=1}^{t} d(k)$ into two pieces,

$$Z(t) \quad \leq \quad \sum_{k=1}^{t} d(k) \quad \leq \quad \left(\sum_{k=1}^{\lfloor (d/\delta)^{1/p} \rfloor} dk^{-p} \right) + \sum_{k=\lfloor (d/\delta)^{1/p} \rfloor + 1}^{t} \delta. \tag{3.19}$$

We want to prove that this implies $Z(t) \leq C_p + \delta t$. The second summand in Inequality (3.19) is always less than δt. The first summand can be bounded above by $d\left(1 + \int_1^{(d/\delta)^{1/p}} x^{-p} \mathrm{d}x\right)$ using a right-hand integral rule. To proceed from here, we must bound this integral with the quantity C_p defined in Theorem 3.3.1. Observe that, because the function x^{-p} is strictly positive on the set $(1, +\infty)$, and because $(d/\delta)^{1/p} \leq d^{1/p}$, we have that $\int_1^{(d/\delta)^{1/p}} x^{-p} \mathrm{d}x \leq \int_1^{d^{1/p}} x^{-p} \mathrm{d}x$. Straight-forward evaluation of this integral (for $p = 1$ and for $p \neq 1$) gives exactly the quantities used to define C_p, and so completes the proof that $Z(t) \leq C_p + \delta t$. Lastly we remark that this analysis comes in handy in the next chapter, in which we generalize our present analysis to apply to the matrix exponential.

4. LOCALIZATION IN THE MATRIX EXPONENTIAL

4.1 Adapting Gauss Southwell for the matrix exponential

In Section 3.3 we studied localization in personalized PageRank by considering the convergence properties of the Gauss Southwell method. This was possible because the personalized PageRank vector is defined as a the solution of a linear system, and the Gauss Southwell method is a linear solver. In this section we construct a framework by which we can view a column of the matrix exponential, $\exp\{\mathbf{P}\}\mathbf{e}_c$, as the solution to a linear system that we derive. Then we demonstrate how to apply Gauss Southwell to our derived system, and consider the convergence of Gauss Southwell in this setting to study the localization of the matrix exponential on graphs. Recall from the previous chapter that we are concerned with a type of strong localization, i.e. a bound on how sparse an approximation $\hat{\mathbf{x}} \approx \exp\{\mathbf{P}\}\mathbf{e}_c$ can be such that it satisfies $\|\exp\{\mathbf{P}\}\mathbf{e}_c - \hat{\mathbf{x}}\|_1 \leq \varepsilon$.

In the remainder of this section we present analysis showing that the framework we construct yields approximations that attain the desired accuracy. In Section 4.2 we study the convergence of the Gauss Southwell method using our framework for a general input, and finally in Section 4.3 we again consider graphs with a skewed degree distribution much like the power-law degree distribution property commonly found in real-world networks. We remark that, just as in Chapter 3, the localization results in this chapter apply to both directed and undirected graphs.

Although in this thesis we propose multiple algorithms for approximating the matrix exponential, in this section we present only one algorithm (which we call `gexpm`) because it is our main tool in proving localization properties for the matrix exponential. We propose our other algorithms and experimental evaluation in Chapter 5. The other algorithms we propose make use of some of the analysis in this section. These methods we first proposed in our paper [Gleich, 2015b], though the analysis given here has been slightly refined since our original development.

The matrix exponential in network analysis. Recently, the matrix exponential has frequently appeared as a tool in the network analysis literature. It has been used to estimate node centrality [Estrada, 2000, Farahat et al., 2006, Estrada and Higham, 2010], for link-prediction [Kunegis and Lommatzsch, 2009], in graph kernels [Kondor and Lafferty, 2002], and – as already mentioned – clustering and community detection [Chung, 2007a]. Many of these studies involve fast ways to approximate the *entire* matrix exponential, instead of a single column as we study here. For instance, Sui et al. [Sui et al., 2013] describe a low-parameter decomposition of a network that is useful both for estimating Katz scores [Katz, 1953] and the matrix exponential. Orecchia and Mahoney [Orecchia

and Mahoney, 2011] show that the heat kernel diffusion implicitly approximates a diffusion operator using a particular type of generalized entropy, which provides a principled rationale for its use.

4.1.1 Taylor Polynomial Approximations of the Exponential

The Taylor series for the exponential of a matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ is given by

$$\exp\{\mathbf{A}\} = \mathbf{I} + \tfrac{1}{1!}\mathbf{A}^1 + \tfrac{1}{2!}\mathbf{A}^2 + \cdots + \tfrac{1}{k!}\mathbf{A}^k + \cdots$$

and it converges for any square matrix $\mathbf{A}$. By truncating this infinite series to N terms, we may define

$$T_N(\mathbf{A}) := \sum_{j=0}^{N} \tfrac{1}{j!}\mathbf{A}^j,$$

and then approximate $\exp\{\mathbf{A}\}\,\mathbf{b} \approx T_N(\mathbf{A})\mathbf{b}$. For general $\mathbf{A}$ this polynomial approximation can lead to inaccurate computations if $\|\mathbf{A}\|$ is large and $\mathbf{A}$ has oppositely signed entries, as the terms $\mathbf{A}^j$ can then contain large, oppositely-signed entries that cancel only in exact arithmetic. However, our aim is to compute $\exp\{\mathbf{P}\}\,\mathbf{e}_c$ specifically for a matrix of bounded norm, that is, $\|\mathbf{P}\|_1 \le 1$. In this setting, the Taylor polynomial approximation is a reliable and accurate tool. What remains is to choose the degree N to ensure the accuracy of the Taylor approximation makes $\|\exp\{\mathbf{P}\}\,\mathbf{e}_c - T_N(\mathbf{P})\mathbf{e}_c\|$ as small as desired.

Choosing the Taylor polynomial degree Accuracy of the Taylor polynomial approximation requires a sufficiently large Taylor degree, N. On the other hand, using a large N requires the algorithms to perform more work. A sufficient value of N can be obtained algorithmically by exactly computing the number of terms of the Taylor polynomial required to compute exp(1) with accuracy ε. Formally:

$$N = \arg\min_k \left\{ k \text{ where } \left(e - \sum_{\ell=0}^{k} \frac{1}{\ell!} \right) \le \varepsilon \right\}.$$

We provide the following simple upper bound on N:

Lemma 4.1.1 *Let $\boldsymbol{P}$ and $\boldsymbol{b}$ satisfy $\|\boldsymbol{P}\|_1, \|\boldsymbol{b}\|_1 \le 1$. Then choosing the degree, N, of the Taylor approximation, $T_N(\boldsymbol{P})$, such that $N \ge 2\log(1/\varepsilon)$ and $N \ge 3$ will guarantee*

$$\|\exp\{\boldsymbol{P}\}\,\boldsymbol{b} - T_N(\boldsymbol{P})\boldsymbol{b}\|_1 \le \varepsilon$$

Because Lemma 4.1.1 provides only a loose bound, we display in Table 4.1 values of N determined via explicit computation of exp(1), which are tight in the case that $\|\mathbf{P}\|_1 = 1$. The values for N in the table show that the bound from Lemma 4.1.1 is not tight, but that both methods for choosing N yield values of N that grow slowly with ε. Next, we prove the lemma.

Proof We first show that the degree N Taylor approximation satisfies

$$\| \exp\{\mathbf{P}\}\,\mathbf{b} - T_N(\mathbf{P})\mathbf{b}\|_1 \leq \tfrac{1}{N!N}. \tag{4.1}$$

To prove this, observe that our approximation's remainder, $\| \exp\{\mathbf{P}\}\,\mathbf{b} - T_N(\mathbf{P})\mathbf{b}\|_1$, equals $\| \sum_{k=N+1}^{\infty} \mathbf{P}^k \mathbf{e}_c/k!\|_1$. Using the triangle inequality we can upperbound this by $\sum_{k=N+1}^{\infty} \|\mathbf{P}^k\|_1 \|\mathbf{e}_c\|_1/k!$.We then have

$$\| \exp\{\mathbf{P}\}\,\mathbf{b} - T_N(\mathbf{P})\mathbf{b}\|_1 \leq \sum_{k=N+1}^{\infty} \tfrac{1}{k!}$$

because $\|\mathbf{P}\|_1 \leq 1$ and $\|\mathbf{e}_c\|_1 = 1$. By factoring out $1/(N+1)!$ and majorizing $(N+1)!/(N+1+k)! \leq 1/(N+1)^k$ for $k \geq 0$, we finish:

$$\| \exp\{\mathbf{P}\}\,\mathbf{b} - T_N(\mathbf{P})\mathbf{b}\|_1 \leq \left(\tfrac{1}{(N+1)!}\right) \sum_{k=0}^{\infty} \left(\tfrac{1}{N+1}\right)^k = \tfrac{1}{(N+1)!}\tfrac{N+1}{N} \tag{4.2}$$

where the last step substitutes the limit for the convergent geometric series.

Next, we prove the lemma. We will show that $2\log(N!) > N\log N$, then use this to relate $\log(N!N)$ to $\log(\varepsilon)$. First we write $2\log(N!) = 2 \cdot \sum_{k=0}^{N-1} \log(1+k) = \sum_{k=0}^{N-1} \log(1+k) + \sum_{k=0}^{N-1} \log(1+k)$. By noting that $\sum_{k=0}^{N-1} \log(1+k) = \sum_{k=0}^{N-1} \log(N-k)$, we can express $2\log(N!) = \sum_{k=0}^{N-1} \log(k+1) + \sum_{k=0}^{N-1} \log(N-k)$, which is equal to $\sum_{k=0}^{N-1} \log\left((k+1)(N-k)\right)$. Finally, $(k+1)(N-k) = N + Nk - k^2 - k = N + k(N-k-1) \geq N$ because $N \geq k+1$, and so

$$2\log(N!) \geq \sum_{k=0}^{N-1} \log(N) = N\log(N). \tag{4.3}$$

By the first claim we know that $1/N!N < \varepsilon$ guarantees the error we want, but for this inequality to hold it is sufficient to have $\log(N!N) > \log(1/\varepsilon)$. Certainly if $\log(N!) > \log(1/\varepsilon)$ then $\log(N!N) > \log(1/\varepsilon)$ holds, so by (4.3) it suffices to choose N satisfying $N\log(N) > 2\log(1/\varepsilon)$. Finally, for $N \geq 3$ we have $\log(N) > 1$, and so Lemma 4.1.1 holds for $N \geq 3$. ■

Table 4.1.
Degree of Taylor polynomial required to approximate the matrix exponential with the desired accuracy.

ε desired	N predicted by Lemma 4.1.1	N required
10^{-5}	24	8
10^{-10}	46	13
10^{-15}	70	17

4.1.2 Error from Approximating the Taylor Approximation

The methods we present in Section 5.1 produce an approximation of the Taylor polynomial expression $T_N(\mathbf{P})\mathbf{e}_c$, which itself approximates $\exp\{\mathbf{P}\}\mathbf{e}_c$. Thus, a secondary error is introduced. Let $\mathbf{x}$ be our approximation of $T_N(\mathbf{P})\mathbf{e}_c$. We find

$$\|\exp\{\mathbf{P}\}\mathbf{e}_c - \mathbf{x}\| \leq \|\exp\{\mathbf{P}\}\mathbf{e}_c - T_N(\mathbf{P})\mathbf{e}_c\| + \|T_N(\mathbf{P})\mathbf{e}_c - \mathbf{x}\|,$$

by the triangle inequality. Lemma 4.1.1 guarantees the accuracy of only the first term; so if the total error of our final approximation $\mathbf{x}$ is to satisfy $\|\exp\{\mathbf{P}\}\mathbf{e}_c - \mathbf{x}\|_1 \leq \varepsilon$, then we must guarantee that the right-hand summand is less than ε. More precisely, we want to ensure for some $\theta \in (0,1)$ that the Taylor polynomial satisfies $\|\exp\{\mathbf{P}\}\mathbf{e}_c - T_N(\mathbf{P})\mathbf{e}_c\|_1 \leq \theta\varepsilon$ and, additionally, our computed approximation $\mathbf{x}$ satisfies $\|T_N(\mathbf{P})\mathbf{e}_c - \mathbf{x}\|_1 \leq (1-\theta)\varepsilon$. We pick $\theta = 1/2$, although we suspect there is an opportunity to optimize this term.

The sections thus far have enabled us to approximate the transcendental function e^x to a desired accuracy via a Taylor polynomial that can be computed using a finite number of matrix vector products. In the next section we show how we can adapt the Gauss Southwell linear solver to accomplish.

4.1.3 Forming a Linear System

We stated a coordinate relaxation method on a linear system. Thus, to use it, we require a linear system whose solution is an approximation of $\exp\{\mathbf{P}\}\mathbf{e}_c$. Here we derive such a system using a Taylor polynomial for the matrix exponential. We present the construction for a general matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ because the Taylor polynomial, linear system, and iterative updates are all well-defined for any real square matrix $\mathbf{A}$; it is only the convergence results that require the additional assumption that $\mathbf{A}$ is a graph-related matrix $\mathbf{P}$ satisfying $\|\mathbf{P}\|_1 \leq 1$.

Consider the product of the degree N Taylor polynomial with $\mathbf{e}_c$:

$$T_N(\mathbf{A})\mathbf{e}_c = \sum_{j=0}^{N} \tfrac{1}{j!}\mathbf{A}^j\mathbf{e}_c \approx \exp\{\mathbf{A}\}\mathbf{e}_c$$

and denote the jth term of the sum by $\mathbf{v}_j := \mathbf{A}^j \mathbf{e}_c / j!$. Then $\mathbf{v}_0 = \mathbf{e}_c$, and the later terms satisfy the recursive relation $\mathbf{v}_{j+1} = \mathbf{A}\mathbf{v}_j/(j+1)$ for $j = 0, ..., N-1$. This recurrence implies that the vectors $\mathbf{v}_j$ satisfy the system

$$\begin{bmatrix} \mathbf{I} & & & & \\ -\mathbf{A}/1 & \mathbf{I} & & & \\ & -\mathbf{A}/2 & \ddots & & \\ & & \ddots & \mathbf{I} & \\ & & & -\mathbf{A}/N & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \vdots \\ \vdots \\ \mathbf{v}_N \end{bmatrix} = \begin{bmatrix} \mathbf{e}_c \\ 0 \\ \vdots \\ \vdots \\ 0 \end{bmatrix}. \tag{4.4}$$

If $\hat{\mathbf{v}} = [\hat{\mathbf{v}}_0, ..., \hat{\mathbf{v}}_N]^T$ is an approximate solution to equation 4.4, then we have $\hat{\mathbf{v}}_j \approx \mathbf{v}_j$ for each term, and so $\sum_{j=0}^{N} \hat{\mathbf{v}}_j \approx \sum_{j=0}^{N} \mathbf{v}_j = T_N(\mathbf{A})\mathbf{e}_c$. Hence, an approximate solution of this linear system yields an approximation of $\exp\{\mathbf{A}\}\,\mathbf{e}_c$. Because the end-goal is computing $\mathbf{x} := \sum_{j=0}^{N} \hat{\mathbf{v}}_j$, we need not form the blocks $\hat{\mathbf{v}}_j$; instead, all updates that would be made to a block of $\hat{\mathbf{v}}$ are instead made directly to $\mathbf{x}$.

We denote the block matrix by $\mathbf{M}$ for convenience; note that the explicit matrix can be expressed more compactly as $(\mathbf{I}_{N+1} \otimes \mathbf{I}_n - \mathbf{S} \otimes \mathbf{A})$, where $\mathbf{S}$ denotes the $(N+1) \times (N+1)$ matrix with first sub-diagonal equal to $[1/1, 1/2, ..., 1/N]$, and $\mathbf{I}_k$ denotes the $k \times k$ identity matrix. Additionally, the right-hand side $[\mathbf{e}_c, 0, ..., 0]^T$ equals $\mathbf{e}_1 \otimes \mathbf{e}_c$. When we apply an iterative method to this system, we often consider sections of the matrix $\mathbf{M} = (\mathbf{I} \otimes \mathbf{I} - \mathbf{S} \otimes \mathbf{A})$, solution $\hat{\mathbf{v}} = [\hat{\mathbf{v}}_0, ..., \hat{\mathbf{v}}_N]^T$, and residual $\mathbf{r} = [\mathbf{r}_0, ..., \mathbf{r}_N]^T$ partitioned into blocks. These vectors each consist of $N+1$ blocks of length n, while $\mathbf{M}$ is an $(N+1) \times (N+1)$ block matrix, with blocks of size $n \times n$.

In practice, this large linear system is never formed, and we work with it implicitly. That is, when the algorithms `gexpm` and `gexpmq` apply coordinate relaxation to the linear system (4.4), we will restate the iterative updates of each linear solver in terms of these blocks. We describe how this can be done efficiently for each algorithm below.

4.1.4 Weighting the Residual Blocks

Before presenting the algorithms, it is necessary to develop some understanding of the error introduced using the linear system in (4.4) approximately. Our goal is to show that the error vector arising from using this system's solution to approximate $T_N(\mathbf{P})$ is a weighted sum of the residual blocks $\mathbf{r}_j$. This is important here because then we can use the coefficients of $\mathbf{r}_j$ to determine the terminating criterion in the algorithms. To begin our error analysis, we look at the inverse of the matrix $\mathbf{M}$.

Lemma 4.1.2 *Let* $\boldsymbol{M} = (\boldsymbol{I}_{N+1} \otimes \boldsymbol{I}_n - \boldsymbol{S} \otimes \boldsymbol{A})$*, where* $\boldsymbol{S}$ *denotes the* $(N+1) \times (N+1)$ *matrix with first sub-diagonal equal to* $[1/1, 1/2, ..., 1/N]$*, and* $\boldsymbol{I}_k$ *denotes the* $k \times k$ *identity matrix. Then* $\boldsymbol{M}^{-1} = \sum_{k=0}^{N} \boldsymbol{S}^k \otimes \boldsymbol{A}^k$.

Proof Because $\mathbf{S}$ is a subdiagonal matrix, it is nilpotent, with $\mathbf{S}^{N+1} = 0$. This implies that $\mathbf{S} \otimes \mathbf{A}$ is also nilpotent, since $(\mathbf{S} \otimes \mathbf{A})^{N+1} = \mathbf{S}^{N+1} \otimes \mathbf{A}^{N+1} = 0 \otimes \mathbf{A} = 0$. Thus, we have

$$\begin{aligned} \mathbf{M}\left(\sum_{k=0}^{N} \mathbf{S}^k \otimes \mathbf{A}^k\right) &= (\mathbf{I} - \mathbf{S} \otimes \mathbf{A})\left(\sum_{k=0}^{N} \mathbf{S}^k \otimes \mathbf{A}^k\right) \\ &= \mathbf{I} - (\mathbf{S} \otimes \mathbf{A})^{N+1} \qquad \text{the sum telescopes} \end{aligned}$$

which is $\mathbf{I}$. This proves $(\sum_{k=0}^{N} \mathbf{S}^k \otimes \mathbf{A}^k)$ is the inverse of $\mathbf{M}$. ■

Next we use the inverse of $\mathbf{M}$ to define our error vector in terms of the residual blocks from the linear system in Section 4.1.3. In order to do so, we need to define a family of polynomials associated with the degree N Taylor polynomial for e^x:

$$\psi_j(x) := \sum_{m=0}^{N-j} \frac{j!}{(j+m)!} x^m \tag{4.5}$$

for $j = 0, 1, ..., N$. Note that these are merely slightly altered truncations of the well-studied functions $\phi_j(x) = \sum_{m=0}^{\infty} \frac{x^m}{(m+j)!}$ that arise in exponential integrators, a class of methods for solving initial value problems. These polynomials $\psi_j(x)$ enable us to derive a precise relationship between the error of the polynomial approximation and the residual blocks of the linear system $\mathbf{Mv} = \mathbf{e}_1 \otimes \mathbf{e}_c$ as expressed in the following lemma.

Lemma 4.1.3 *Consider an approximate solution* $\hat{\boldsymbol{v}} = [\hat{\boldsymbol{v}}_0; \hat{\boldsymbol{v}}_1; \cdots; \hat{\boldsymbol{v}}_N]$ *to the linear system*

$$(\boldsymbol{I}_{N+1} \otimes \boldsymbol{I}_n - \boldsymbol{S} \otimes \boldsymbol{A})[\boldsymbol{v}_0; \boldsymbol{v}_1; \cdots; \boldsymbol{v}_N] = \boldsymbol{e}_1 \otimes \boldsymbol{e}_c.$$

Let $\boldsymbol{x} = \sum_{j=0}^{N} \hat{\boldsymbol{v}}_j$*, let* $T_N(x)$ *be the degree* N *Taylor polynomial for* e^x*, and define* $\psi_j(x) = \sum_{m=0}^{N-j} \frac{j!}{(j+m)!} x^m$*. Define the residual vector* $\boldsymbol{r} = [\boldsymbol{r}_0; \boldsymbol{r}_1; \ldots; \boldsymbol{r}_N]$ *by* $\boldsymbol{r} := \boldsymbol{e}_1 \otimes \boldsymbol{e}_c - (\boldsymbol{I}_{N+1} \otimes \boldsymbol{I}_n - \boldsymbol{S} \otimes \boldsymbol{A})\hat{\boldsymbol{v}}$*. Then the error vector* $T_N(\boldsymbol{A})\boldsymbol{e}_c - \boldsymbol{x}$ *can be expressed*

$$T_N(\boldsymbol{A})\boldsymbol{e}_c - \boldsymbol{x} = \sum_{j=0}^{N} \psi_j(\boldsymbol{A})\boldsymbol{r}_j.$$

The essence of the proof is that, using Lemma 4.1.2, we can write a simple formulation for $\mathbf{M}^{-1}\mathbf{r}$, which is the expression for the error. Here we present the proof in full detail.

Proof Recall that $\mathbf{v} = [\mathbf{v}_0; \mathbf{v}_1; \cdots; \mathbf{v}_N]$ is the solution to equation (4.4), and our approximation is $\hat{\mathbf{v}} = [\hat{\mathbf{v}}_0; \hat{\mathbf{v}}_1; \cdots; \hat{\mathbf{v}}_N]$. We showed in Section 4.1.3 that the error $T_N(\mathbf{A})\mathbf{e}_c - \mathbf{x}$ is in fact the sum of

the error blocks $\mathbf{v}_j - \hat{\mathbf{v}}_j$. Now we will express the error blocks $\mathbf{v}_j - \hat{\mathbf{v}}_j$ in terms of the residual blocks of the system (4.4), i.e. $\mathbf{r}_j$.

The following relationship between the residual vector and solution vector always holds: $\mathbf{r} = \mathbf{e}_1 \otimes \mathbf{e}_c - \mathbf{M}\hat{\mathbf{v}}$, so pre-multiplying by $\mathbf{M}^{-1}$ yields $\mathbf{M}^{-1}\mathbf{r} = \mathbf{v} - \hat{\mathbf{v}}$, because $\mathbf{v} = \mathbf{M}^{-1}\mathbf{e}_1 \otimes \mathbf{e}_c$ exactly, by definition of $\mathbf{v}$. Note that $\mathbf{M}^{-1}\mathbf{r} = \mathbf{v} - \hat{\mathbf{v}}$ is the error vector for the linear system (4.4). Substituting the expression for $\mathbf{M}^{-1}$ in Lemma 4.1.2 yields

$$\begin{bmatrix} \mathbf{v}_0 - \hat{\mathbf{v}}_0 \\ \mathbf{v}_1 - \hat{\mathbf{v}}_1 \\ \vdots \\ \mathbf{v}_N - \hat{\mathbf{v}}_N \end{bmatrix} = \left(\sum_{k=0}^{N} \mathbf{S}^k \otimes \mathbf{A}^k \right) \begin{bmatrix} \mathbf{r}_0 \\ \mathbf{r}_1 \\ \vdots \\ \mathbf{r}_N \end{bmatrix}. \tag{4.6}$$

Let $\mathbf{e}$ be the vector of all 1s of appropriate dimension. Then observe that pre-multiplying equation (4.6) by $\left(\mathbf{e}^T \otimes \mathbf{I}\right)$ yields, on the left-hand side, $\sum_{j=0}^{N}(\mathbf{v}_j - \hat{\mathbf{v}}_j)$. Now we can accomplish our goal of expressing $\sum_{j=0}^{N}(\mathbf{v}_j - \hat{\mathbf{v}}_j)$ in terms of the residual blocks $\mathbf{r}_j$ by expressing the right-hand side $\left(\mathbf{e}^T \otimes \mathbf{I}\right)\left(\sum_{k=0}^{N} \mathbf{S}^k \otimes \mathbf{A}^k\right)\mathbf{r}$ in terms of the blocks $\mathbf{r}_j$. So next we consider the product of a fixed block $\mathbf{r}_{j-1}$ with a particular term $(\mathbf{S}^k \otimes \mathbf{A}^k)$. Note that, because $\mathbf{r}_{j-1}$ is in block-row j of $\mathbf{r}$, it multiplies with only the block-column j of $(\mathbf{S}^k \otimes \mathbf{A}^k)$, so we now examine the blocks in block-column j of $(\mathbf{S}^k \otimes \mathbf{A}^k)$.

Because $\mathbf{S}$ is a subdiagonal matrix, there is only one non-zero in each column of $\mathbf{S}^k$, for each $k = 0, ..., N$. As mentioned in Section 4.1.5, $\mathbf{S}\mathbf{e}_j = \mathbf{e}_{j+1}/j$ when $j < N+1$, and 0 otherwise. This implies that

$$\mathbf{S}^k\mathbf{e}_j = \begin{cases} \frac{(j-1)!}{(j-1+k)!}\mathbf{e}_{j+k}, & \text{if } 0 \le k \le N+1-j \\ 0, & \text{otherwise.} \end{cases}$$

Thus, block-column j of $(\mathbf{S}^k \otimes \mathbf{A}^k)$ contains only a single non-zero block, $(j-1)!\mathbf{A}^k/(j-1+k)!$, for each $k = 0, ..., N+1-j$. Hence, summing the $n \times n$ blocks in block-column j of all powers $(\mathbf{S}^k \otimes \mathbf{A}^k)$ for $k = 0, ..., N$ yields

$$\sum_{k=0}^{N+1-j} \frac{(j-1)!}{(j-1+k)!}\mathbf{A}^k \tag{4.7}$$

as the matrix coefficient of the term $\mathbf{r}_{j-1}$ in the expression $(\mathbf{e}^T \otimes \mathbf{I})(\sum_{k=0}^{N} \mathbf{S}^k \otimes \mathbf{A}^k)\mathbf{r}$. Thus, we have

$$\left(\mathbf{e}^T \otimes \mathbf{I}\right)\left(\sum_{k=0}^{N} \mathbf{S}^k \otimes \mathbf{A}^k\right)\mathbf{r} = \sum_{j=1}^{N+1}\left(\sum_{k=0}^{N+1-j} \frac{(j-1)!}{(j-1+k)!)}\mathbf{A}^k\right)\mathbf{r}_{j-1}$$

Finally, reindexing so that the outer summation on the right-hand side goes from $j = 0$ to N, then substituting our definition for $\psi_j(\mathbf{A}) = \sum_{k=0}^{N-m} \frac{j!}{(j+k)!}\mathbf{A}^k$, we have that $\sum_{j=0}^{N}(\mathbf{v}_j - \hat{\mathbf{v}}_j) = \sum_{j=0}^{N} \psi_j(\mathbf{A})\mathbf{r}_j$, as desired. ■

4.1.5 Approximating the Taylor Polynomial via Gauss-Southwell

The main idea of `gexpm`, our first algorithm, is to apply Gauss-Southwell to the system (4.4) in a way that exploits the sparsity of both $\mathbf{M}$ and the input matrix $\mathbf{P}$. In particular, we need to adapt the coordinate and residual updates of Gauss-Southwell in (2.2) for the system (4.4) by taking advantage of the block structure of the system.

We begin our iteration to solve $\mathbf{Mv} = \mathbf{e}_1 \otimes \mathbf{e}_c$ with $\hat{\mathbf{x}}jv0 = 0$ and $\mathbf{r}^{(0)} = \mathbf{e}_1 \otimes \mathbf{e}_c$. Consider an approximate solution after k steps of Gauss-Southwell, $\hat{\mathbf{v}}^{(k)}$, and residual $\mathbf{r}^{(k)}$. The standard GS iteration consists of adding the largest entry of $\mathbf{r}^{(k)}$, call it $m^{(k)} := \mathbf{r}_q^{(k)}$, to $\hat{\mathbf{v}}_q^{(k)}$, and then updating $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - m^{(k)}\mathbf{M}\mathbf{e}_q$.

We want to rephrase the iteration using the block structure of our system. We will denote the jth block of $\mathbf{r}$ by $\mathbf{r}_{j-1}$, and entry q of $\mathbf{r}$ by $(\mathbf{r})_q$. Note that the entry q corresponds with node i in block $j-1$ of the residual, $\mathbf{r}_{j-1}$. Thus, if the largest entry is $(\mathbf{r}^{(k)})_q$, then we write $\mathbf{e}_q = \mathbf{e}_j \otimes \mathbf{e}_i$ and the largest entry in the residual is $m^{(k)} := (\mathbf{e}_j \otimes \mathbf{e}_i)^T\mathbf{r}^{(k)} = \mathbf{e}_i^T\mathbf{r}_{j-1}^{(k)}$. The standard GS update to the solution would then add $m^{(k)}(\mathbf{e}_j \otimes \mathbf{e}_i)$ to the iterative solution, $\hat{\mathbf{v}}^{(k)}$; but this simplifies to adding $m^{(k)}\mathbf{e}_i$ to block $j-1$ of $\hat{\mathbf{v}}^{(k)}$, i.e. $\hat{\mathbf{v}}_{j-1}^{(k)}$. In practice we never form the blocks of $\hat{\mathbf{v}}$, and instead simply add $m^{(k)}\mathbf{e}_i$ to $\mathbf{x}^{(k)}$, our iterative approximation of $\exp\{\mathbf{P}\}\mathbf{e}_c$.

The standard update to the residual is $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - m^{(k)}\mathbf{M}\mathbf{e}_q$. Using the block notation and expanding $\mathbf{M} = \mathbf{I}\otimes\mathbf{I} - \mathbf{S}\otimes\mathbf{P}$, the residual update becomes $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - m^{(k)}\mathbf{e}_j \otimes \mathbf{e}_i + (\mathbf{S}\mathbf{e}_j)\otimes(\mathbf{P}\mathbf{e}_i)$. Furthermore, we can simplify the product $\mathbf{S}\mathbf{e}_j$ using the structure of $\mathbf{S}$: for $j = 1, ..., N$, we have $\mathbf{S}\mathbf{e}_j = \mathbf{e}_{j+1}/j$; if $j = N+1$, then $\mathbf{S}\mathbf{e}_j = 0$.

To implement this iteration, we needed q, the index of the largest entry of the residual vector. To ensure this operation is fast, we store the residual vector's non-zero entries in a heap. This allows $O(1)$ lookup time for the largest magnitude entry each step at the cost of reheaping the residual each time an entry of $\mathbf{r}$ is altered.

We want the algorithm to terminate once its 1-norm error is below a prescribed tolerance, ε. To ensure this, we maintain a weighted sum of the 1-norms of the residual blocks, $t^{(k)} = \sum_{j=0}^{N} \psi_j(1)\|\mathbf{r}_j^{(k)}\|_1$. Now we can reduce the entire `gexpm` iteration to the following:

1. Set $m^{(k)} = (\mathbf{e}_j \otimes \mathbf{e}_i)^T\mathbf{r}^{(k)}$, the top entry of the heap, then delete the entry in $\mathbf{r}^{(k)}$ so that $(\mathbf{e}_j \otimes \mathbf{e}_i)^T\mathbf{r}^{(k+1)} = 0$.
2. Update $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + m^{(k)}\mathbf{e}_i$.
3. If $j < N+1$, update $\mathbf{r}_j^{(k+1)} = \mathbf{r}_j^{(k)} + m^{(k)}\mathbf{P}\mathbf{e}_i/j$, reheaping $\mathbf{r}$ after each add.
4. Update $t^{(k+1)} = t^{(k)} - \psi_{j-1}(1)|m^{(k)}| + \psi_j(1)|m^{(k)}|/j$.

We show in Theorem 4.2.1 that iterating until $t^{(k)} \leq \varepsilon$ guarantees a 1-norm accuracy of ε.

4.2 Convergence of Gauss Southwell

We divide our theoretical analysis into two stages. In the first we establish the convergence of our adapted Gauss Southwell, `gexpm`, for a class of matrices that includes column-stochastic matrices. Then, in Section 4.3, we give improved convergence results for our method when the underlying graph has a degree distribution that has a skewed degree distribution similar to a power-law, which we define formally in Section 4.3. In the remainder of this section we show that `gexpm` converges to an approximate solution with a prescribed 1-norm error ε for any matrix $\mathbf{P}$ satisfying $\|\mathbf{P}\|_1 \leq 1$.

Consider the large linear system (4.4) using a matrix $\mathbf{P}$ with 1-norm bounded by one. Then by applying both Lemma 4.1.3 and the triangle inequality, we find that the error in approximately solving the system can be expressed in terms of the residuals in each block:

$$\|T_N(\mathbf{P})\mathbf{e}_c - \mathbf{x}\|_1 \leq \sum_{j=0}^{N} \|\psi_j(\mathbf{P})\|_1 \|\mathbf{r}_j\|_1.$$

Because the polynomials $\psi_j(t)$ have all nonnegative coefficients, and because $\psi_j(\mathbf{P})$ is a polynomial in $\mathbf{P}$ for each j, we have that $\|\psi_j(\mathbf{P})\|_1 \leq \psi_j(\|\mathbf{P}\|_1)$. Finally, using the condition that $\|\mathbf{P}\|_1 \leq 1$, we have proved the following:

Lemma 4.2.1 *Consider the setting from Lemma 4.1.3 applied to a matrix $\|\boldsymbol{P}\|_1 \leq 1$. Then the norm of the error vector $T_N(\boldsymbol{A})\boldsymbol{e}_c - \boldsymbol{x}$ associated with an approximate solution is a weighted sum of the residual norms from each block:*

$$\|T_N(\boldsymbol{P})\boldsymbol{e}_c - \boldsymbol{x}\|_1 \leq \sum_{j=0}^{N} \psi_j(1)\|\boldsymbol{r}_j\|_1.$$

Note that this does not require nonnegativity of either $\mathbf{r}$ or $\mathbf{P}$, only that $\|\mathbf{P}\|_1 \leq 1$; this improves on our original analysis in [Kloster and Gleich, 2013].

We now show that our algorithm monotonically decreases the weighted sum of residual norms, and hence converges to a solution. The intuition is that each relaxation step reduces the residual in block j and increases the residual in block $j+1$, but by a smaller amount. Thus, the relaxation steps monotonically reduce the residuals.

Theorem 4.2.1 *Let $\boldsymbol{P} \in \mathbb{R}^{n\times n}$ satisfy $\|\boldsymbol{P}\|_1 \leq 1$. Then in the notation of Section 4.1.5, the residual vector after l steps of `gexpm` satisfies $\|\boldsymbol{r}^{(l)}\|_1 \leq l^{-1/(2d)}$ and the error vector satisfies*

$$\|T_N(\boldsymbol{P})\boldsymbol{e}_c - \boldsymbol{x}\|_1 \leq \exp(1) \cdot l^{\left(-\frac{1}{2d}\right)}, \tag{4.8}$$

so `gexpm` converges in at most $l = (\exp(1)/\varepsilon)^{2d}$ iterations.

Proof The iterative update described in Section 4.1.5 involves a residual block, say $\mathbf{r}_{j-1}$, and a row index, say i, so that the largest entry in the residual at step l is $m^{(l)} = (\mathbf{e}_j \otimes \mathbf{e}_i)^T\mathbf{r}^{(l)}$. First, the

residual is updated by deleting the value $m^{(l)}$ from entry i of the block $\mathbf{r}_{j-1}^{(l)}$, which results in the 1-norm of the residual decreasing by exactly $|m^{(l)}|$. Then, we add $m^{(l)}\mathbf{P}\mathbf{e}_i/j$ to $\mathbf{r}_j^{(l)}$, which results in the 1-norm of the residual increasing by at most $\|m^{(l)}\mathbf{P}\mathbf{e}_i/j\|_1 \leq |m^{(l)}/j|$, since $\|\mathbf{P}\mathbf{e}_i\|_1 \leq 1$. Thus, the net change in the 1-norm of the residual will satisfy

$$\|\mathbf{r}^{(l+1)}\|_1 \leq \|\mathbf{r}^{(l)}\| - |m^{(l)}| + \left|\tfrac{m^{(l)}}{j}\right|.$$

Note that the first residual block, $\mathbf{r}_0$, has only a single non-zero in it, since $\mathbf{r}_0 = \mathbf{e}_c$ in the initial residual. This means that every step after the first operates on residual $\mathbf{r}_{j-1}$ for $j \geq 2$. Thus, for every step after step 0, we have that $1/j \leq 1/2$. Hence, we have

$$\|\mathbf{r}^{(l+1)}\|_1 \leq \|\mathbf{r}^{(l)}\| - |m^{(l)}| + \left|\tfrac{m^{(l)}}{2}\right| = \|\mathbf{r}^{(l)}\| - \tfrac{|m^{(l)}|}{2}.$$

We can lowerbound $|m^{(l)}|$, the largest-magnitude entry in the residual, with the average magnitude of the residual. The average value of $\mathbf{r}$ equals $\|\mathbf{r}\|_1$ divided by the number of non-zeros in $\mathbf{r}$. After l steps, the residual can have no more than dl non-zero elements, since at most d non-zeros can be introduced in the residual each time $\mathbf{P}\mathbf{e}_i$ is added; hence, the average value at step l is lowerbounded by $\|\mathbf{r}^{(l)}\|_1/dl$. Substituting this into the previous inequality, we have

$$\|\mathbf{r}^{(l+1)}\|_1 \leq \|\mathbf{r}^{(l)}\| - \tfrac{|m^{(l)}|}{2} \leq \|\mathbf{r}^{(l)}\| - \tfrac{\|\mathbf{r}^{(l)}\|_1}{2dl} = \|\mathbf{r}^{(l)}\|\left(1 - \tfrac{1}{2dl}\right).$$

Iterating this inequality yields the bound $\|\mathbf{r}^{(l)}\|_1 \leq \|\mathbf{r}^{(0)}\|_1 \prod_{k=1}^{l}(1-1/(2dk))$, and since $\mathbf{r}^{(0)} = \mathbf{e}_1 \otimes \mathbf{e}_c$ we have $\|\mathbf{r}^{(0)}\|_1 = 1$. Thus, $\|\mathbf{r}^{(l)}\|_1 \leq \prod_{k=1}^{l}(1 - 1/(2dk))$. The first inequality of (4.2.1) follows from using the facts $(1+x) \leq e^x$ (for $x > -1$) and $\log(l) < \sum_{k=1}^{l} 1/k$ to write

$$\prod_{k=1}^{l}(1 - \tfrac{1}{2dk}) \leq \exp\left\{-\tfrac{1}{2d}\sum_{k=1}^{l}\tfrac{1}{k}\right\} \leq \exp\left\{-\tfrac{1}{2d}\log l\right\} = l^{\left(-\frac{1}{2d}\right)}.$$

The inequality $(1+x) \leq e^x$ follows from the Taylor series $e^x = 1 + x + o(x^2)$, and the lowerbound for the partial harmonic sum $\sum_{k=1}^{l} 1/k$ follows from the left-hand rule integral approximation $\log(l) = \int_1^l (1/x)\,dx < \sum_{k=1}^{l} 1/k$.

Finally, to prove inequality (4.8), we use the fact that $\psi_j(1) \leq \psi_0(1) \leq \exp(1)$ for all $j = 0, ..., N$. First, we justify this claim. By definition, $\psi_j(1) = \sum_{m=0}^{N-j} \frac{j!}{(j+m)!}$ and $\psi_{j+1}(1) = \sum_{m=0}^{N-j-1} \frac{(j+1)!}{(j+1+m)!}$. Note that the sum for $\psi_j(1)$ has more terms, and the general terms of the two summations satisfy $\frac{j!}{(j+m)!} \geq \frac{(j+1)!}{(j+1+m)!}$ because multiplying both sides by $\frac{(j+m)!}{j!}$ yields $1 \geq \frac{j+1}{j+1+m}$. Hence $\psi_j(1) \geq \psi_{j+1}(1)$ for $j = 0, ..., N-1$, and so the statement follows.

To see that $\psi_0(1) \leq \exp(1)$, note that $\psi_0(1)$ is the degree N Taylor polynomial expression for $\exp(1)$, which is a finite approximation of the Taylor series, an infinite sum of positive terms; hence, $\psi_0(1) \leq \exp(1)$.

Thus, we have $\|T_N(\mathbf{P})\mathbf{e}_c - \mathbf{x}\|_1 \leq \psi_0(1) \sum_{j=0}^{N} \|\mathbf{r}_j\|_1$ by Lemma 4.2.1. Next, note that $\sum_{j=0}^{N} \|\mathbf{r}_j\|_1 = \|\mathbf{r}\|_1$, because $\mathbf{r} = [\mathbf{r}_0, \mathbf{r}_1, ..., \mathbf{r}_N]^T$. Combining these facts we have $\|T_N(\mathbf{P})\mathbf{e}_c - \mathbf{x}\|_1 \leq \exp(1)\|\mathbf{r}\|_1$, which proves the error bound. The bound on the number of iterations required for convergence follows from simplifying the inequality $\exp(1)l^{-1/(2d)} < \varepsilon$. ■

We remark that for matrices with maximum number of nonzeros per column bounded by $\log\log n$, this result proves that `gexpm` converges in a sublinear amount of work. This applies to banded matrices with a bandwidth of $\log\log n$ or less.

4.3 Using the Skewed Degree Distribution

In our convergence analysis for `gexpm` in Section 4.2, the inequalities rely on our estimation of the largest entry in the residual vector at step l, $m^{(l)}$. In this section we achieve a tighter bound on $m^{(l)}$ by using the distribution of the degrees of the underlying graph instead of just d, the maximum degree. In the case that the degree sequence follows the particular kind of skewed distribution we first explained in Section 3.3.3, we show that the improvement on the bound on $m^{(l)}$ leads to a sublinear runtime for the algorithm.

Recall from Section 3.3.3 that we make the following assumptions about the degree sequence. The graph has a skewed degree distribution much like a power-law degree distribution, meaning that the kth largest degree satisfies $d(k) \leq d \cdot k^{-p}$ for a constant p in $(0.5, 1]$, and where $d = d(1)$ is the maximum degree in the graph. Let δ denote the minimum degree of the graph and note that, for sparse networks in which $|E| = O(n)$, δ is a small constant (which, realistically, is $\delta = 1$ in real-world networks). A degree distribution of this kind applies to a variety of real-world networks [Faloutsos et al., 1999a]. A more commonly-used definition states that the number of nodes having degree k is equal to k^{-a}, but the two definitions can be shown to be equivalent for a certain range of values of their respective exponents, p and a [Adamic, 2002]. In this definition, the values of the exponent a for real-world networks range from 2 to 3, frequently closer to 2. These values correspond to $p = 1$ (for $a = 2$) and $p = 1/2$ (for $a = 3$) in the definition that we use. Finally, we note that, though the definition that we use contains an equality, our results hold for any graph with a degree distribution satisfying a "sub" power-law, meaning $d(k) \leq d \cdot k^{-p}$. We now state our main result, then establish some preliminary technical lemmas before finally proving it.

Theorem 4.3.1 *For a graph with degree distribution following a power-law with $p \in (0.5, 1]$, max degree d, and minimum degree δ, `gexpm` converges to a 1-norm error of ε in work bounded by*

$$work(\varepsilon) = \begin{cases} O\left(\log(\frac{1}{\varepsilon})\left(\frac{1}{\varepsilon}\right)^{\frac{3\delta}{2}} d^2 \log(d) \max\{\log(d), \log(\frac{1}{\varepsilon})\}\right) & \text{if } p = 1 \\ O\left(\log(\frac{1}{\varepsilon})\left(\frac{1}{\varepsilon}\right)^{\frac{3\delta}{2}} d^2 \frac{1}{1-p}\left(d^{\frac{1}{p}-1} - 1\right) \max\{\log(d), \log(\frac{1}{\varepsilon})\}\right) & \text{if } p \neq 1 \end{cases} \tag{4.9}$$

Note that when the maximum degree satisfies $d < n^r$ for any $r < 1/(1 + 1/p)$, and the minimum degree is a constant independent of n, Theorem 4.3.1 implies that the runtime scales sublinearly with the graph size, for a fixed 1-norm error of ε.

In practice, having a minimum degree that is a small constant independent of n is extremely common, and values of p are typically near or slightly less than 1. The condition on the maximum degree (that $d < n^r$ for $r < 1/(1 + 1/p)$) is slightly less common, with five of our seven datasets (listed in Table 5.1) satisfying $d < 2.5 \cdot n^{1/2}$.

4.3.1 Bounding the Number of Non-zeros in the Residual

In the proof of Theorem 4.2.1 we showed that the residual update satisfies $\|\mathbf{r}^{(l+1)}\|_1 \leq \|\mathbf{r}^{(l)}\|_1 - m^{(l)}(1 - 1/j)$, where $m^{(l)}$ is the largest entry in $\mathbf{r}^{(l)}$, and $\mathbf{r}_{j-1}$ is the section of the residual vector where the entry $m^{(l)}$ is located. We used the bound $m^{(l)} \geq \|\mathbf{r}^{(l)}\|_1/(dl)$, which is a lowerbound on the average value of all entries in $\mathbf{r}^{(l)}$. This follows from the loose upperbound dl on the number of non-zeros in $\mathbf{r}^{(l)}$. We also used the naive upperbound $1/2$ on $(1 - 1/j)$. Here we prove new bounds on these quantities. For the sake of simpler expressions in the proofs, we express the number of iterations as a multiple of N, i.e. Nl.

Lemma 4.3.1 *Let $d(k)$:= the kth largest degree in the graph (with repetition), let $Z(m) := \sum_{k=1}^{m} d(k)$, and let $nnz(l)$:= the number of non-zero entries in $\mathbf{r}^{(l)}$. Then after Nl iterations of* `gexpm` *we have*

$$nnz(Nl) \leq N \cdot Z(l). \tag{4.10}$$

Proof At any given step, the number of new non-zeros we can create in the residual vector is bounded above by the largest degree of all the nodes which have not already had their neighborhoods added to $\mathbf{r}^{(Nl)}$. If we have already explored the node with degree $= d(1)$, then the next node we introduce to the residual cannot add more than $d(2)$ new non-zeros to the residual, because the locations in $\mathbf{r}$ in which the node $d(1)$ would create non-zeros already have non-zero value.

We cannot conclude $\text{nnz}(l) \leq \sum_{k=1}^{l} d(k) = Z(l)$ because this ignores the fact that the same set of $d(1)$ nodes can be introduced into each different time step of the residual, $j = 2, \cdots, N$. Recall that entries of the residual are of the form $\mathbf{e}_j \otimes \mathbf{e}_i$ where i is the index of the node, $i = 1, \cdots, n$; and j is the section of the residual, or time step: $j = 2, \cdots, N$ (note that j skips 1 because the first iteration of GS deletes the only entry in section $j = 1$ of the residual). Recall that the entry of $\mathbf{r}$ corresponding to the 1 in the vector $\mathbf{e}_j \otimes \mathbf{e}_i$ is located in block $\mathbf{r}_{j-1}$. Then for each degree, $d(1), d(2), \cdots, d(l)$, we

have to add non-zeros to that set of $d(k)$ nodes in each of the $N-1$ different blocks $\mathbf{r}_{j-1}$ before we move on to the next degree, $d(l+1)$:

$$\begin{aligned}\text{nnz}(Nl) &\leq d(1)+\cdots+d(1)+d(2)+\cdots+d(2)+\cdots+d(l)\\ &\leq Nd(1)+Nd(2)+\cdots+Nd(l)\end{aligned}$$

which equals $N\cdot\sum_{k=1}^{l}d(k)=N\cdot Z(l)$. ■

Lemma 4.3.1 enables us to rewrite the inequality $-m_{Nl}\leq-\|\mathbf{r}^{(Nl)}\|/(dNl)$ from the proof of Theorem 4.2.1 as $-m_{Nl}\leq-\|\mathbf{r}^{(Nl)}\|/(N\cdot Z(l))$. Letting σ_{Nl} represent the value of $(1-1/j)$ in step Nl, we can write our upperbound on $\|\mathbf{r}^{(Nl+1)}\|_1$ as follows:

$$\|\mathbf{r}^{(Nl+1)}\|_1\leq\|\mathbf{r}^{(Nl)}\|_1\left(1-\tfrac{\sigma_{Nl}}{N\cdot Z(l)}\right). \tag{4.11}$$

We want to recur this by substituting a similar inequality in for $\|\mathbf{r}^{(Nl)}\|_1$, but the indexing does not work out because inequality (4.10) holds only when the number of iterations is of the form Nl. We can overcome this by combining N iterations into one expression:

Lemma 4.3.2 *In the notation of Lemma 4.3.1, let* $s_{l+1}:=\min\{\sigma_{Nl+N},\sigma_{Nl+N-1},\cdots,\sigma_{Nl+1}\}$. *Then the residual from* Nl *iterations of* `gexpm` *satisfies*

$$\|\boldsymbol{r}^{(N(l+1))}\|_1\leq\|\boldsymbol{r}^{(Nl)}\|_1\left(1-\tfrac{s_{l+1}}{N\cdot Z(l+1)}\right)^N. \tag{4.12}$$

Proof By Lemma 4.3.1 we know that $N\cdot Z(l)\geq\text{nnz}(Nl)$ for all l. In the proof of Lemma 4.3.1 we showed that, during step Nl, no more than $d(l)$ new non-zeros can be created in the residual vector. By the same argument, no more than $d(l+1)$ non-zeros can be created in the residual vector during steps $Nl+k$, for $k=1,...,N$. Thus we have $\text{nnz}(Nl+k)\leq N\cdot Z(l)+k\cdot d(l+1)\leq N\cdot Z(l)+Nd(l+1)=N\cdot Z(l+1)$ for $k=0,1,...,N$. With this we can bound $-m_{Nl+k}\leq-\|\mathbf{r}^{(Nl+k)}\|_1/(N\cdot Z(l+1))$ for $k=0,...,N$. Recall we defined σ_{Nl} to be the value of $(1-1/j)$ in step Nl. With this in mind, we establish a new bound on the residual decrease at each step:

$$\begin{aligned}\|\mathbf{r}^{(N(l+1))}\|_1 &\leq\|\mathbf{r}^{(Nl+N-1)}\|_1-m_{Nl+N-1}\sigma_{Nl+N-1}\\ &\leq\|\mathbf{r}^{(Nl+N-1)}\|_1-\tfrac{\|\mathbf{r}^{(Nl+N-1)}\|_1\sigma_{Nl+N-1}}{N\cdot Z(l+1)}\\ &=\|\mathbf{r}^{(Nl+N-1)}\|_1\left(1-\tfrac{\sigma_{Nl+N-1}}{N\cdot Z(l+1)}\right)\\ &\leq\left(\|\mathbf{r}^{(Nl+N-2)}\|_1-m_{Nl+N-2}\sigma_{Nl+N-2}\right)\left(1-\tfrac{\sigma_{Nl+N-1}}{N\cdot Z(l+1)}\right)\\ &\leq\|\mathbf{r}^{(Nl+N-2)}\|_1\left(1-\tfrac{\sigma_{Nl+N-2}}{N\cdot Z(l+1)}\right)\left(1-\tfrac{\sigma_{Nl+N-1}}{N\cdot Z(l+1)}\right),\end{aligned}$$

and recurring this yields

$$\|\mathbf{r}^{(N(l+1))}\|_1\leq\|\mathbf{r}^{(Nl)}\|_1\prod_{t=1}^{N}\left(1-\tfrac{\sigma_{Nl+N-t}}{N\cdot Z(l+1)}\right).$$

From the definition of s_{l+1} in the statement of the lemma, we can upperbound $-\sigma_{Nl+N-t}$ in the last inequality with $-s_{l+1}$. This enables us to replace the product in the last inequality with $(1 - s_{l+1}/(N \cdot Z(l+1)))^N$, which proves the lemma. ■

Recurring the inequality in (4.12) bounds the residual norm in terms of $Z(m)$:

Corollary 4.3.1 *In the notation of Lemma 4.3.2, after Nl iterations of* `gexpm` *the residual satisfies*

$$\|\boldsymbol{r}^{(Nl)}\|_1 \leq \exp\left\{-\sum_{k=1}^{l} \tfrac{s_k}{Z(k)}\right\} \tag{4.13}$$

Proof By recurring the inequality of Lemma 4.3.2, we establish the new bound $\|\mathbf{r}^{(Nl)}\|_1 \leq \|\mathbf{r}^{(0)}\|_1 \prod_{k=1}^{l} (1 - s_k/(N \cdot Z(k)))^N$. The factor $(1 - s_k/(N \cdot Z(k)))^N$ can be upperbounded by $\exp\left\{\sum_{k=1}^{l} (-s_k/(N \cdot Z$ using the inequality $1 - x \leq \exp(-x)$. Cancelling the factors of N and noting that $\|\mathbf{r}^{(0)}\|_1 = 1$ completes the proof. ■

We want an upperbound on $-\sum_{k=1}^{m} 1/Z(k)$, so we need a lowerbound on $\sum_{k=1}^{m} 1/Z(k)$. This requires a lowerbound on $1/Z(k)$, which in turn requires an upperbound on $Z(k)$. So next we upperbound $Z(k)$ using the degree distribution, which ultimately will allow us to upperbound $\|\mathbf{r}^{(Nl)}\|_1$ by an expression of d, the max degree.

4.3.2 Skewed Degree Distributions

If we assume that the graph has a skewed degree distribution much like a power-law degree distribution, then we can bound $d(k) \leq d \cdot k^{-p}$ for a constants $p \in (0.5, 1]$. Let δ denote the minimum degree of the graph and note that, for sparse networks in which $|E| = O(n)$, δ is a small constant (which, realistically, is $\delta = 1$ in real-world networks). With these bounds in place, we will bound $Z(k)$ in terms of d, δ, and p.

We remark that this is essentially the same setting that we use in Section 3.3 to demonstrate localization in personalized PageRank. For the convenience of the reader, we restate here Lemma 3.3.1 which we proved in our earlier analysis, but will re-use here. Recalling our definition of the constant C_p

$$C_p = \begin{cases} d(1 + \log d) & \text{if } p = 1 \\ d\left(1 + \frac{1}{(1-p)}\left(d^{\frac{1}{p}-1} - 1\right)\right) & \text{if } p \in (0,1)\ . \end{cases} \tag{4.14}$$

Lemma 3.3.1 states that $Z(k) \leq C_p + \delta k$. We want to use this tighter bound on $Z(k)$ to establish a tighter bound on $\|\mathbf{r}^{(Nl)}\|_1$. We can accomplish this using inequality (4.13) if we first bound the sum $\sum_{k=b}^{m} 1/Z(k)$ for constants b, m.

Lemma 4.3.3 *In the notation of Section 4.3.2 we have*

$$\sum_{k=b}^{m} \tfrac{1}{Z(k)} \geq \tfrac{1}{\delta} \log\left(\frac{\delta m + \delta + C_p}{\delta b + C_p}\right) \tag{4.15}$$

Proof From Lemma 3.3.1 we can write $Z(k) \leq C_p + \delta k$. Then $1/Z(k) \geq 1/(C_p + \delta k)$, and so we have $\sum_{k=1}^{m} 1/Z(k) \geq \sum_{k=1}^{m} 1/(C_p + \delta k)$. Using a left-hand rule integral approximation, we get

$$\sum_{k=b}^{m} \tfrac{1}{C_p+\delta k} \geq \int_{b}^{m+1} \tfrac{1}{C_p+\delta x} \mathrm{d}x = \tfrac{1}{\delta} \log\left(\frac{\delta m + \delta + C_p}{\delta b + C_p}\right). \tag{4.16}$$

■

Plugging (4.15) into (4.13) yields, after some manipulation, our sublinearity result:

Theorem 4.3.2 *In the notation of Section 4.3.2, for a graph with a skewed degree distribution with decay exponent $p \in (0, 1]$, `gexpm` attains $\|\boldsymbol{r}^{(Nl)}\|_1 < \varepsilon$ in Nl iterations if $l > (3/\delta)(1/\varepsilon)^{3\delta/2} C_p$.*

Proof Before we can substitute (4.15) into (4.13), we have to control the coefficients s_i. Note that the only entries in $\mathbf{r}$ for which $s_k = (1 - 1/j)$ is equal to $1/2$ are the entries that correspond to the earliest time step, $j = 2$ (in the notation of Section 4.1.3). There are at most d iterations that have a time step value of $j = 2$, because only the neighbors of the starting node, node c, have non-zero entries in the $j = 2$ time step. Hence, every iteration other than those d iterations must have $s_k \geq (1 - 1/j)$ with $j \geq 3$, which implies $s_k \geq \frac{2}{3}$. We cannot say *which* d iterations of the Nl total iterations occur in time step $j = 2$. However, the first d values of $1/Z(k)$ in $\sum_{k=1}^{m} s_k/Z(k)$ are the largest in the sum, so by assuming those d terms have the smaller coefficient ($1/2$ instead of $2/3$), we can guarantee that

$$-\sum_{k=1}^{l} \tfrac{s_k}{Z(k)} < -\sum_{k=1}^{d} \tfrac{1/2}{Z(k)} - \sum_{k=d+1}^{l} \tfrac{2/3}{Z(k)} \tag{4.17}$$

To make the proof simpler, we omit the sum $\sum_{k=1}^{d}(1/2)/Z(k)$ outright. From Corollary 4.3.1 we have $\|\mathbf{r}^{(Nl)}\|_1 \leq \exp\left\{-\sum_{k=1}^{l} s_k/Z(k)\right\}$, which we can bound above with $\exp\left\{-(2/3)\sum_{k=d+1}^{l} 1/Z(k)\right\}$, using inequality (4.17). Lemma 4.3.3 allows us to upperbound the sum $-\sum_{k=d+1}^{l} 1/Z(k)$, and simplifying yields

$$\|\mathbf{r}^{(Nl)}\|_1 \leq \left(\frac{\delta l + \delta + C_p}{\delta(d+1) + C_p}\right)^{-\frac{2}{3\delta}}.$$

To guarantee $\|\mathbf{r}^{(Nl)}\|_1 < \varepsilon$, then, it suffices to show that $\delta l + \delta + C_p > (1/\varepsilon)^{3\delta/2}(\delta d + \delta + C_p)$. This inequality holds if l is greater than $(1/\delta)(1/\varepsilon)^{3\delta/2}(\delta d + \delta + C_p)$. Hence, it is enough for l to satisfy

$$l \geq \tfrac{3}{\delta}(\tfrac{1}{\varepsilon})^{\frac{3\delta}{2}} C_p.$$

This last line requires the assumption $(\delta d + \delta + C_p) < 3C_p$, which holds only if $\log d$ is larger than δ (in the case $p = 1$), or if $d^{\frac{1}{p}-1}$ is larger than δ (in the case $p \neq 1$). Since we have been assuming that d is a function of n and δ is a constant independent of n, it is safe to assume this. ■

With these technical lemmas in place, we are prepared to prove Theorem 4.3.1 that gives the runtime bound for the `gexpm` algorithm on graphs with a skewed degree distribution.

Proof of Theorem 4.3.1 Theorem 4.3.2 states that $l \geq (3/\delta)(1/\varepsilon)^{3\delta/2}C_p$ will guarantee $\|\mathbf{r}^{(Nl)}\|_1 < \varepsilon$. It remains to count the number of floating point operations performed in Nl iterations.

Each iteration involves a vector add consisting of at most d operations, and adding a column of $\mathbf{P}$ to the residual, which consists of at most d adds. Then, each entry that is added to the residual requires a heap update. The heap updates at iteration k involve at most $O(\log \text{nnz}(k))$ work, since the residual heap contains at most $\text{nnz}(k)$ non-zeros at that iteration. The heap is largest at the last iteration, so we can upperbound $\text{nnz}(k) \leq \text{nnz}(Nl)$ for all $k \leq Nl$. Thus, each iteration consists of no more than d heap updates, and so $d \log(\text{nnz}(Nl))$ total operations involved in updating the heap. Hence, after Nl iterations, the total amount of work performed is upperbounded by $O(Nld \log \text{nnz}(Nl))$.

After applying Lemmas 4.3.1 and 3.3.1 we know the number of non-zeros in the residual (after Nl iterations) will satisfy $\text{nnz}(Nl) \leq N \cdot Z(l) < N\,(C_p + \delta l)$. Substituting in the expression for l from 4.3.2 yields $\text{nnz}(Nl) < N\left(C_p + \delta(3/\delta)(1/\varepsilon)^{3\delta/2}C_p\right)$. Upperbounding $C_p < (1/\varepsilon)^{3\delta/2}C_p$ allows us to write

$$\text{nnz}(Nl) < 4N(\tfrac{1}{\varepsilon})^{\frac{3\delta}{2}}C_p. \tag{4.18}$$

We can upperbound the work, $\text{work}(\varepsilon)$, required to produce a solution with error $< \varepsilon$, by using the inequalities in (4.18) and in Theorem 4.3.2. We expand the bound $\text{work}(\varepsilon) < Nld \log \text{nnz}(Nl)$ to

$$\begin{aligned}
&< Nd\left((\tfrac{3}{\delta})(\tfrac{1}{\varepsilon})^{\frac{3\delta}{2}}C_p\right) \cdot \log(4N(\tfrac{1}{\varepsilon})^{\frac{3\delta}{2}}C_p) \\
&< N\left((\tfrac{3}{\delta})(\tfrac{1}{\varepsilon})^{\frac{3\delta}{2}}dC_p\right) \cdot \left(\log(4N) + \frac{3\delta}{2}\log(\tfrac{1}{\varepsilon}) + \log(C_p)\right),
\end{aligned}$$

which we can upperbound with $O\left(3N(\frac{1}{\varepsilon})^{\frac{3\delta}{2}}dC_p \cdot 4 \cdot \max\{\log(d), \log(1/\varepsilon)\}\right)$. This proves $\text{work}(\varepsilon) = O\left(N\,(1/\varepsilon)^{3\delta/2}\,dC_p \cdot \max\{\log(C_p), \log(1/\varepsilon)\}\right)$. Replacing N with the expression from Lemma 4.1.1 yields the bound on total work given in Theorem 4.3.1.

This completes our localization results for the matrix exponential, and our theoretical analysis of our `gexpm` algorithm. We give experimental analysis of `gexpm` in the following chapter, where we introduce our other algorithms for computing the matrix exponential.

5. ALGORITHMS AND ROUNDED MATRIX-VECTOR PRODUCTS

In Chapter 4 we presented our novel adaptation of the Gauss-Southwell linear solver to compute a vector times the exponential of a matrix, which we called `gexpm`. In this chapter we present modifications of `gexpm` designed to be even faster on large networks (Section 5.1) and present experimental analysis demonstrating a significant speed-up of our algorithms of the state of the art (Section 5.2). In essence each of these methods can be understood as performing rounded matrix-vector products to reduce the work and fill-in incurred during repeated matrix vectors products carried out in evaluating a vector times a Taylor polynomial of a matrix. With this perspective in mind, in Chapter 6 we consider the more general task of how to optimally round a vector to minimize the work in a subsequent matrix-vector product.

5.1 Fast algorithms for the matrix exponential

In this section we present two more algorithms for approximating $\exp\{\mathbf{P}\}\,\mathbf{e}_c$. The methods apply most efficiently to random walk transition matrices for sparse graphs, but they work on any nonnegative matrix with $\|\mathbf{P}\|_1 \leq 1$. The methods consist of coordinate relaxation steps on a linear system, $\mathbf{M}$, that we construct from a Taylor polynomial approximating $\exp\{\mathbf{P}\}$, as explained in Section 4.1.3. The first algorithm is a close relative of the algorithm `gexpm` presented in the previous section, but it stores *significant entries* of the residual in a queue rather than maintaining a heap as `gexpm` does. This makes it faster, and also turns out to be closely related to a truncated Gauss-Seidel method. Because of the queue, we call this second method `gexpmq`.

The second algorithm approximates the product $T_N(\mathbf{P})\mathbf{e}_c$ using Horner's rule on the polynomial $T_N(\mathbf{P})$, in concert with a procedure we call an "incomplete" matrix-vector product (Section 5.1.2). This procedure deletes all but the largest entries in the vector before performing a matrix-vector product.

We re-use much of the analysis from 4.1 in developing the two algorithms for the matrix exponential in this section, in particular much of the convergence analysis from Section 4.2. We construct `gexpmq` such that the solutions they produce have guaranteed accuracy. On the other hand, `expmimv` sacrifices predictable accuracy for a guaranteed fast runtime bound.

5.1.1 Approximating the Taylor Polynomial via Gauss-Seidel

Here we describe an algorithm similar to our `gexpm` in the previous chapter, that stores the residual in a queue to avoid the heap updates of `gexpm`. Our original inspiration for this method was the relationship between the Bookmark Coloring Algorithm [Berkhin, 2007] and the Push method for Personalized PageRank [Andersen et al., 2006a]. The rationale for this change is that maintaining the heap in `gexpm` is slow. Remarkably, the final algorithm we create is actually a version of Gauss-Seidel that *skips* updates from insignificant residuals, whereas standard Gauss-Seidel cycles through coordinates of the matrix cyclically in index order. Our algorithm will use the queue to do *one* such pass and maintain *significant entries* of the residual that must be relaxed (and not skipped).

The basic iterative step is the same as in `gexpm`, except that the entry of the residual chosen, say $(\mathbf{e}_j \otimes \mathbf{e}_i)^T\mathbf{r}$, is not selected to be the largest in $\mathbf{r}$. Instead, it is the next entry in a queue storing significant entries of the residual. Then as entries in $\mathbf{r}$ are updated, we place them at the back of the queue, Q. Note that the block-wise nature of our update has the following property: an update from the jth block results in residuals changing in the $(j+1)$st block. Because new elements are added to the tail of the queue, all entries of $\mathbf{r}_{j-1}$ are relaxed before proceeding to $\mathbf{r}_j$.

If carried out exactly as described, this would be equivalent to performing each product $\mathbf{v}_j = \mathbf{P}\mathbf{v}_{j-1}/j$ in its entirety. But we want to avoid these full products; so we introduce a rounding threshold for determining whether or not to operate on the entries of the residual as we pop them off of Q.

The rounding threshold is determined as follows. After every entry in $\mathbf{r}_{j-1}$ is removed from the top of Q, then all entries remaining in Q are in block $\mathbf{r}_j$ (remember, this is because operating on entries in $\mathbf{r}_{j-1}$ adds to Q only entries that are from $\mathbf{r}_j$.) Once every entry in $\mathbf{r}_{j-1}$ is removed from Q, we set $Z_j = |Q|$, the number of entries in Q; this is equivalent to the total number of non-zero entries in $\mathbf{r}_j$ before we begin operating on entries of $\mathbf{r}_j$. Then, while operating on $\mathbf{r}_j$, the threshold used is

$$\text{threshold}(\varepsilon, j, N) = \tfrac{\varepsilon}{N\psi_j(1)Z_j}. \tag{5.1}$$

Then, each step, an entry is popped off of Q, and if it is larger than this threshold, it is operated on; otherwise, it is simply discarded, and the next entry of Q is considered. Once again, we maintain a weighted sum of the 1-norms of the residual blocks, $t^{(k)} = \sum_{j=0}^{N} \psi_j(1)\|\mathbf{r}_j^{(k)}\|_1$, and terminate once $t^{(k)} \leq \varepsilon$, or if the queue is empty.

Step $k+1$ of `gexpmq` is as follows:

1. Pop the top entry of Q, call it $r = (\mathbf{e}_j \otimes \mathbf{e}_i)^T\mathbf{r}^{(k)}$, then delete the entry in $\mathbf{r}^{(k)}$, so that $(\mathbf{e}_j \otimes \mathbf{e}_i)^T\mathbf{r}^{(k+1)} = 0$.
2. If $r \geq \text{threshold}(\varepsilon, j, N)$ do the following:

(a) Add $r\mathbf{e}_i$ to $\mathbf{x}_i$.

(b) Add $r\mathbf{P}\mathbf{e}_i/j$ to residual block $\mathbf{r}_j^{(k+1)}$.

(c) For each entry of $\mathbf{r}_j^{(k+1)}$ that was updated, add that entry to the back of Q.

(d) Update $t^{(k+1)} = t^{(k)} - \psi_{j-1}(1)|r| + \psi_j(1)|r|/j$.

We show in the proof of Theorem 5.1.2 that iterating until $t^{(k)} \leq \varepsilon$, or until all entries in the queue satisfying the threshold condition have been removed, will guarantee that the resulting vector $\mathbf{x}$ will approximate $\exp\{\mathbf{P}\}\,\mathbf{e}_c$ with the desired accuracy.

5.1.2 A sparse, heuristic approximation

The above algorithms guarantee that the final approximation attains the desired accuracy ε. Here we present an algorithm designed to be faster. Because we have no error analysis for this algorithm currently, and because the steps of the method are well-defined for any $\mathbf{A} \in \mathbb{R}^{n\times n}$, we discuss this algorithm in a more general setting. This method also uses a Taylor polynomial for $\exp\{\mathbf{A}\}$, but does not use the linear system constructed for the previous two methods. Instead, the Taylor terms are computed via Horner's rule on the Taylor polynomial. But, rather than a full matrix-vector product, we apply what we call an"incomplete" matrix-vector product (IMV) to compute the successive terms. Thus, our name: `expmimv`. We describe the IMV procedure before describing the algorithm.

Incomplete Matrix-vector Products (IMV) Given any matrix $\mathbf{A}$ and a vector $\mathbf{v}$ of compatible dimension, the IMV procedure sorts the entries of $\mathbf{v}$, then removes all entries except for the largest z. Let $[\mathbf{v}]_z$ denote the vector $\mathbf{v}$ with all but its z largest-magnitude entries deleted. Then we define the *z-incomplete matrix-vector product* of $\mathbf{A}$ and $\mathbf{v}$ to be $\mathbf{A}[\mathbf{v}]_z$. We call this an *incomplete* product, rather than a rounded matrix-vector product, because, although the procedure is equivalent to rounding to 0 all entries in $\mathbf{v}$ below some threshold, that rounding-threshold is not known a priori, and its value will vary from step to step in our algorithm.

There are likely to be a variety of ways to implement these IMVs. Ours computes $[\mathbf{v}]_z$ by filtering all the entries of $\mathbf{v}$ through a min-heap of size z. For each entry of $\mathbf{v}$, if that entry is larger than the minimum value in the heap, then replace the old minimum value with the new entry and re-heap; otherwise, set that entry in $[\mathbf{v}]_z$ to be zero, then proceed to the next entry of $\mathbf{v}$. Many similar methods have been explored in the literature before, for instance [Yuan and Zhang, 2011].

Horner's rule with IMV A Horner's rule approach to computing $T_N(\mathbf{A})$ considers the polynomial as follows:

$$\begin{aligned}\exp\{\mathbf{A}\} \approx T_N(\mathbf{A}) &= \mathbf{I} + \tfrac{1}{1!}\mathbf{A}^1 + \tfrac{1}{2!}\mathbf{A}^2 + \cdots + \tfrac{1}{N!}\mathbf{A}^N \\ &= \mathbf{I} + \tfrac{1}{1}\mathbf{A}\left(\mathbf{I} + \tfrac{1}{2}\mathbf{A}\left(\mathbf{I} + \cdots + \tfrac{1}{N-1}\mathbf{A}\left(\mathbf{I} + \tfrac{1}{N}\mathbf{A}\right)\cdots\right)\right)\end{aligned} \tag{5.2}$$

Using this representation, we can approximate $\exp\{\mathbf{A}\}\,\mathbf{e}_c$ by multiplying $\mathbf{e}_c$ by the inner-most term, $\mathbf{A}/N$, and working from the inside out. More precisely, the `expmimv` procedure is as follows:

1. Fix $z \in \mathbb{N}$.
2. Set $\mathbf{x}^{(0)} = \mathbf{e}_c$.
3. For $k = 0, ..., N-1$ compute $\mathbf{x}^{(k+1)} = \mathbf{A}\left([\mathbf{x}^{(k)}]_z/(N-k)\right) + \mathbf{e}_c$.

Then at the end of this process we have $\mathbf{x}^{(N)} \approx T_N(\mathbf{A})\mathbf{e}_c$. The vector $[\mathbf{x}^{(k)}]_z$ used in each iteration of step 3 is computed via the IMV procedure described above. For an experimental analysis of the speed and accuracy of `expmimv`, see Section 5.2.1.

Runtime analysis Now assume that the matrix $\mathbf{A}$ in the above presentation corresponds to a graph, and let d be the maximum degree found in the graph related to $\mathbf{A}$. Each step of `expmimv` requires identifying the z largest entries of $\mathbf{v}^{(k)}$, multiplying $\mathbf{A}[\mathbf{v}^{(k)}]_z$, then adding $\mathbf{e}_c$. If $\mathbf{v}$ has $\text{nnz}(\mathbf{v})$ non-zeros, and the largest z entries are desired, then computing $[\mathbf{v}]_z$ requires at most $O(\text{nnz}(\mathbf{v})\log(z))$ work: each of the $\text{nnz}(\mathbf{v})$ entries are put into the size-z heap, and each heap update takes at most $O(\log(z))$ operations.

Note that the number of non-zeros in $\mathbf{v}^{(k)}$, for any k, can be no more than dz. This is because the product $\mathbf{v}^{(k)} = \mathbf{A}[\mathbf{v}^{(k-1)}]_z$ combines exactly z columns of the matrix: the z columns corresponding to the z non-zeros in $[\mathbf{v}^{(k)}]_z$. Since no column of $\mathbf{A}$ has more than d non-zeros, the sum of these z columns can have no more than dz non-zeros. Hence, computing $[\mathbf{v}^{(k)}]_z$ from $\mathbf{v}^{(k)}$ requires at most $O(dz\log(z))$ work. Observe also that the work done in computing the product $\mathbf{A}[\mathbf{v}^{(k)}]_z$ cannot exceed dz. Since exactly N iterations suffice to evaluate the polynomial, we have proved Theorem 5.1.1:

Theorem 5.1.1 *Let $\boldsymbol{A}$ be any graph-related matrix having maximum degree d. Then the `expmimv` procedure, using a heap of size z, computes an approximation of $\exp\{\boldsymbol{A}\}\,\boldsymbol{e}_c$ via an N degree Taylor polynomial in work bounded by $O(Ndz\log z)$.*

If $\mathbf{A}$ satisfies $\|\mathbf{A}\|_1 \leq 1$, then by Lemma 4.1.1 we can choose N to be a small constant to achieve a coarse $O(10^{-3})$ approximation.

While the `expmimv` method always has a sublinear runtime, we currently have no theoretical analysis of its accuracy. However, in our experiments we found that a heap size of $z = 10,000$ yields a 1-norm accuracy of $\approx 10^{-3}$ for social networks with millions of nodes (Section 5.2.1). Yet, even

for a fixed value of z, the accuracy varied widely. For general-purpose computation of the matrix exponential, we do not recommend this procedure. If instead the purpose is identifying large entries of $\exp\{\mathbf{P}\}\,\mathbf{e}_c$, our experiments suggest that `expmimv` often accomplishes this task with high accuracy (Section 5.2.1).

5.1.3 Convergence of Coordinate Relaxation Methods

In this section, we show `gexpmq` converges to an approximate solution with a prescribed 1-norm error ε for any matrix $\mathbf{P}$ satisfying $\|\mathbf{P}\|_1 \leq 1$. The intuition for the convergence analysis is similar to that for `gexpm`: each relaxation step reduces the residual in block j and increases the residual in block $j+1$, but by a smaller amount. Thus, the relaxation steps monotonically reduce the residuals. Here we formally state the convergence result for `gexpmq`.

Theorem 5.1.2 *Let $\boldsymbol{P} \in \mathbb{R}^{n\times n}$ satisfy $\|\boldsymbol{P}\|_1 \leq 1$. Then in the notation of Section 5.1.1, using a threshold of*

$$\textit{threshold}(\varepsilon, j, N) = \tfrac{\varepsilon}{N\psi_j(1)Z_j}$$

for each residual block $\boldsymbol{r}_j$ will guarantee that when `gexpmq` terminates, the error vector satisfies $\|T_N(\boldsymbol{P})\boldsymbol{e}_c - \boldsymbol{x}\|_1 \leq \varepsilon$.

Proof From Lemma 4.2.1 we have $\|T_N(\mathbf{P})\mathbf{e}_c - \mathbf{x}\|_1 \leq \sum_{j=0}^{N} \psi_j(1)\|\mathbf{r}_j\|_1$. During the first iteration we remove the only non-zero entry in $\mathbf{r}_0 = \mathbf{e}_c$ from the queue, then add $\mathbf{P}\mathbf{e}_c$ to $\mathbf{r}_1$. Thus, when the algorithm has terminated, we have $\|\mathbf{r}_0\|_1 = 0$, and so we can ignore the term $\psi_0(1)\|\mathbf{r}_0\|_1$ in the sum. In the other N blocks of the residual, $\mathbf{r}_j$ for $j = 1, ..., N$, the steps of `gexpmq` delete every entry with magnitude r satisfying $r \geq \varepsilon/(N\psi_j(1)Z_j)$. This implies that all entries remaining in block $\mathbf{r}_j$ are bounded above in magnitude by $\varepsilon/(N\psi_j(1)Z_j)$. Since there can be no more than Z_j non-zero entries in $\mathbf{r}_j$ (by definition of Z_j), we have that $\|\mathbf{r}_j\|_1$ is bounded above by $Z_j \cdot \varepsilon/(N\psi_j(1)Z_j)$. Thus, we have

$$\|T_N(\mathbf{P})\mathbf{e}_c - \mathbf{x}\|_1 \leq \sum_{j=1}^{N} \psi_j(1)\left(Z_j \tfrac{\varepsilon}{N\psi_j(1)Z_j}\right)$$

and simplifying completes the proof. ■

Currently we have no theoretical runtime analysis for `gexpmq`. However, because of the algorithm's similarity to `gexpm`, and because of our strong heuristic evidence (presented in Section 5.2), we believe a rigorous theoretical runtime bound exists.

5.2 Experimental Results

Here we evaluate our algorithms' accuracy and speed for large real-world and synthetic networks.

Overview To evaluate accuracy, we examine how well the `gexpmq` function identifies the largest entries of the true solution vector. This is designed to study how well our approximation would work in applications that use large-magnitude entries to find important nodes (Section 5.2.1). We find a tolerance of 10^{-4} is sufficient to accurately find the largest entries at a variety of scales. We also provide more insight into the convergence properties of `expmimv` by measuring the accuracy of the algorithm as the size z of its heap varies (Section 5.2.1). Based on these experiments, we recommend setting the subset size for that algorithm to be near $(\text{nnz}(\mathbf{P})/n)$ times the number of large entries desired.

We then study how the algorithms scale with graph size. We first compare their runtimes on real-world graphs with varying sizes (Section 5.2.2). The edge density and maximum degree of the graph will play an important role in the runtime. This study illustrates a few interesting properties of the runtime that we examine further in an experiment with synthetic forest-fire graphs of up to a billion edges. Here, we find that the runtime scaling grows roughly as d^2, as predicted by our theoretical results.

Real-world networks The datasets used are summarized in Table 5.1, and span three orders of magnitude in size. They include a version of the flickr graph from [Bonchi et al., 2012] containing just the largest strongly-connected component of the original graph; dblp-2010 from [Boldi et al., 2011], itdk0304 in [(The Cooperative Association for Internet Data Analyais), 2005], ljournal-2008 from [Boldi et al., 2011, Chierichetti et al., 2009], twitter-2010 [Kwak et al., 2010] webbase-2001 from [Hirai et al., 2000, Boldi and Vigna, 2005], and the friendster graph in [Yang and Leskovec, 2012].

Table 5.1.
Properties of datasets for our matrix exponential experiments.

Graph	$\lvert V\rvert$	$\text{nnz}(\mathbf{P})$	$\text{nnz}(\mathbf{P})/\lvert V\rvert$	d	$\sqrt{\lvert V\rvert}$
`itdk0304`	190,914	1,215,220	6.37	1,071	437
`dblp-2010`	226,413	1,432,920	6.33	238	476
`flickr-scc`	527,476	9,357,071	17.74	9,967	727
`ljournal-2008`	5,363,260	77,991,514	14.54	2,469	2,316
`webbase-2001`	118,142,155	1,019,903,190	8.63	3,841	10,870
`twitter-2010`	33,479,734	1,394,440,635	41.65	768,552	5,786
`friendster`	65,608,366	3,612,134,270	55.06	5,214	8,100

Implementation details All experiments were performed on either a dual processor Xeon e5-2670 system with 16 cores (total) and 256GB of RAM or a single processor Intel i7-990X, 3.47 GHz CPU and 24 GB of RAM. Our algorithms were implemented in C++ using the Matlab MEX interface. All data structures used are memory-efficient: the solution and residual are stored as hash tables using Google's `sparsehash` package. The precise code for the algorithms and the experiments below are available via `https://www.cs.purdue.edu/homes/dgleich/codes/nexpokit/`.

Comparison We compare our implementation with a state-of-the-art Matlab function for computing the exponential of a matrix times a vector, `expmv`, which uses a Taylor polynomial approach [Al-Mohy and Higham, 2011]. We customized this method with the knowledge that $\|\mathbf{P}\| 1 = 1$. This single change results in a great improvement to the runtime of their code. In each experiment, we use as the "true solution" the result of a call to `expmv` using the 'single' option, which guarantees a relative backward error bounded by 2^{-24}, or, for smaller problems, we use a Taylor approximation with the number of terms predicted by Lemma 3.3.1.

5.2.1 Accuracy on Large Entries

When both `gexpm` and `gexpmq` terminate, they satisfy a 1-norm error of ε. Many applications do not require precise solution *values* but instead would like the correct *set* of large-magnitude entries. To measure the accuracy of our algorithms in identifying these large-magnitude entries, we examine the set precision of the approximations. Recall that the precision of a set T that approximates a desired set S is the size of their intersection divided by the total size: $|S \cap T|/|S|$. Precision values near 1 indicate accurate sets and values near 0 indicate inaccurate sets. We show the precision as we vary the solution tolerance ε for the `gexpmq` method in Figure 5.1. The experiment we conduct is to take a graph, estimate the matrix exponential for 100 vertices (trials) for our method `gexpmq` with various tolerances ε, and compare the sets of the top 100 vertices *that are not neighbors of the seed node* between the true solution and the solution from our algorithm. We remove the starting node and its neighbors because these entries are *always* large, so accurately identifying them is a near-guarantee. The results show that, in median performance computed over 100 trials, we get the top 100 node set completely correct with $\varepsilon = 10^{-4}$ for the small graphs.

Next, we study how the work performed by the algorithm `gexpmq` scales with its precision in identifying the most important nodes. For this study, a single instance of our experiment consists of the following. We pick a vertex at random and vary the maximum number of iterations performed by the `gexpmq` algorithm. Then, we look at the set precision for the top-k sets. The horizontal axis in Figure 5.2 measures the number of effective matrix-vector products based on the number of edges

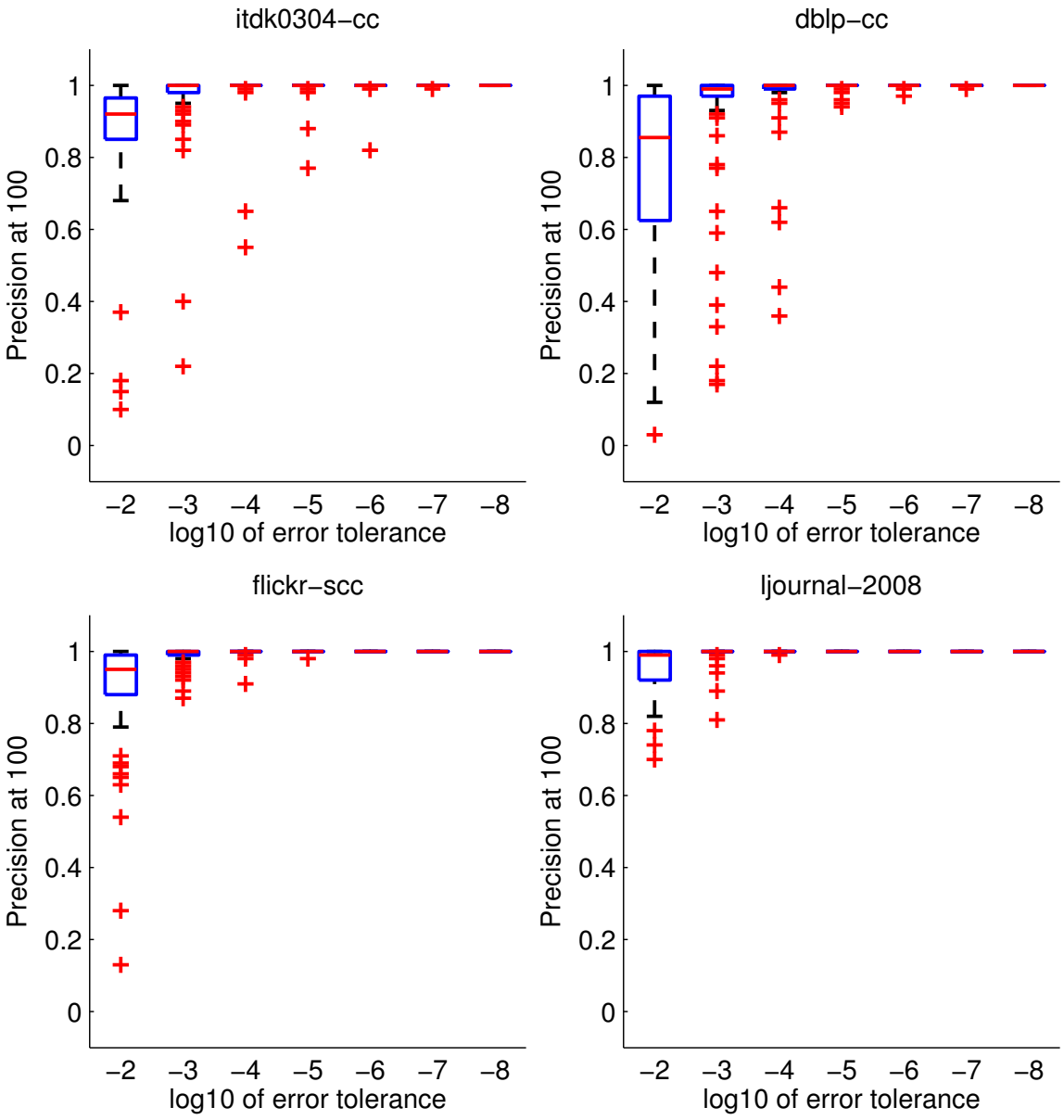

Figure 5.1. Precision on top-k nodes vs error tolerance for `gexpmq`.

explored divided by the total number of non-zeros of the matrix. Thus, one matrix-vector product of work corresponds with looking at each non-zero in the matrix once. The four images in Figure 5.2 report results over 100 trials of this experiment. They show that we get good accuracy for the top-k sets up to $k = 1000$ with a tolerance of 10^{-4}, *and* converge in less than one matrix-vector product, with the sole exception of the flickr network. This network has been problematic for previous studies as well [Bonchi et al., 2012]. Here, we note that we get good results in less than one matrix-vector product, but we do not detect convergence until after a few matrix-vector products worth of work.

Accuracy & non-zeros with incomplete matrix-vector products

The previous studies explored the accuracy of the `gexpmq` method. Our cursory experiments showed that `gexpm` behaves similarly because it also achieves an ε error in the 1-norm. In contrast, the `expmimv` method is rather different in its accuracy because it prescribes only a total size of intermediate heap; we are interested in accuracy as we let the heap size increase.

The precise experiment is as follows. For each graph, repeat the following: first, compute 50 node indices uniformly at random. Each node will serve as the seed node for a single trial of our experiment. For each node index, use `expmimv` to compute $\exp\{\mathbf{P}\}\,\mathbf{e}_c$ using different values for the heap size parameter: $z = 100$, 200, 500, 1000, 2000, 5000, 10000. Figure 5.3 displays the median of these 50 trials for each parameter setting for both the 1-norm error and the top-1000 set precision of the `expmimv` approximations. (The results for the top-100 precision, as in the previous study, were effectively the same.) The left plot shows the 1-norm error compared with the number of non-zeros retained in the matrix-vector products on our set of graphs. This left plot displays clear differences for the various graphs, yet there are pairs of graphs that have very similar errors (such as itdk0304 and dblp).

The common characteristic for each pair (of graphs with similar error curves) appears to be the edge density of the graph. We see this correlation of accuracy with edge-density more strongly in the right plot where we look at the precision in the top-1000 set as as we increase the number of non-zeros that `expmimv` uses, relative to the edge density of the graph ($\mathrm{nnz}(\mathbf{P})/n$). Again, set precision improves for all datasets as more non-zeros are used. If we normalize by edge density (by dividing the number of non-zeros used by the edge density of each graph) then the curves cluster. Once the ratio (non-zeros used / edge density) reaches 100, `expmimv` attains a set precision over 0.95 for *all* datasets on the 1,000 largest-magnitude nodes, regardless of the graph size. We view this as strong evidence that this method should be useful in many applications where precise numeric values are not required.

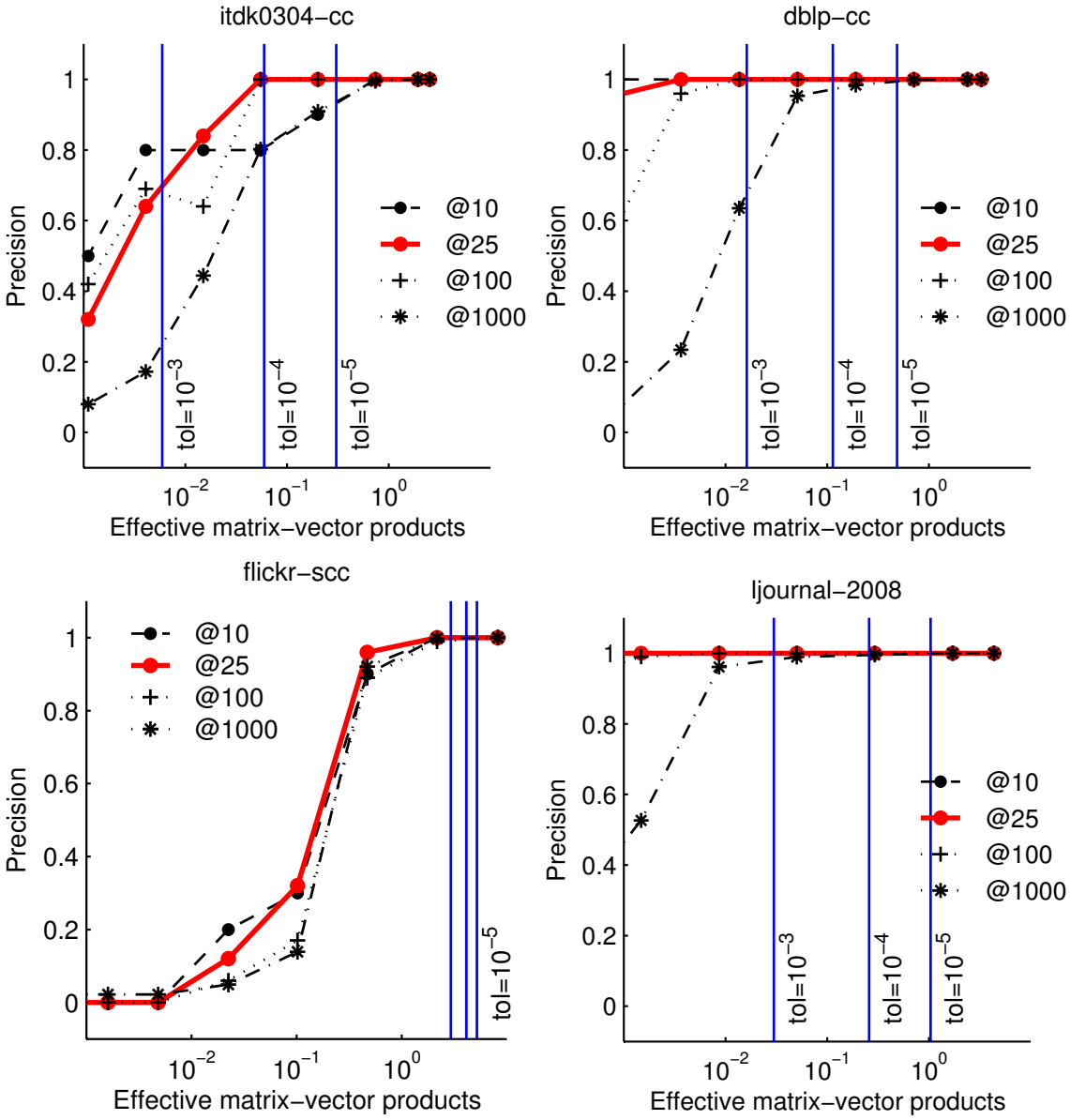

Figure 5.2. Precision on top-k nodes vs work performed by `gexpmq`.

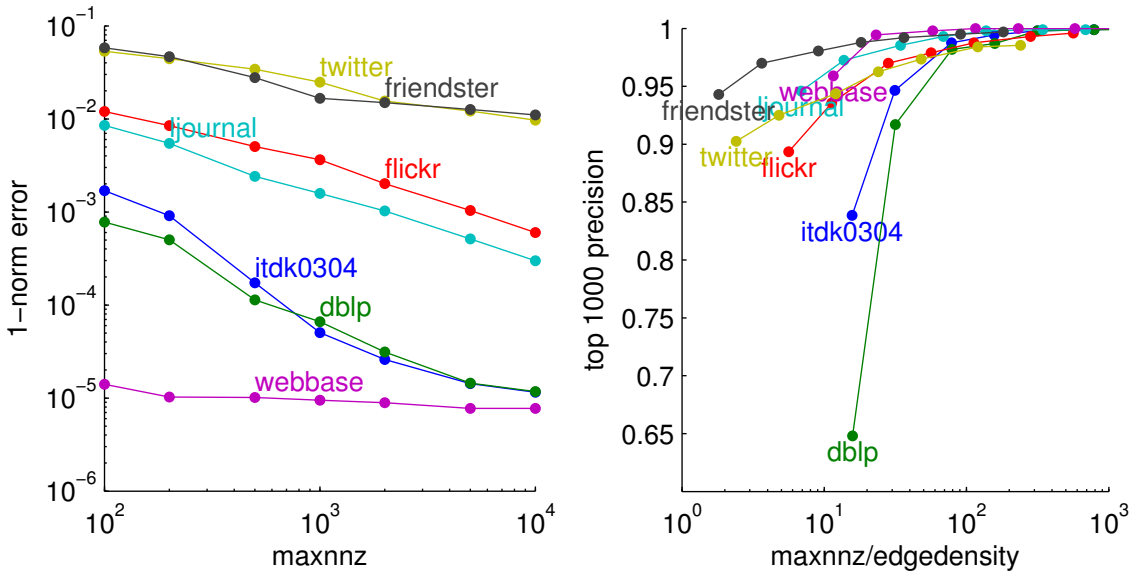

Figure 5.3. Precision on top-k nodes vs set size used in `expmimv`.

5.2.2 Runtime & Input-size

Because the algorithms presented here are intended to be fast on large, sparse networks, we continue our study by investigating how their speed scales with data size. Figure 5.4 displays the median runtime for each graph, where the median is taken over 100 trials for the smaller graphs, and 50 trials for the twitter and friendster datasets. Each trial consists of computing a column $\exp\{\mathbf{P}\}\,\mathbf{e}_c$ for randomly chosen c. All algorithms use a fixed 1-norm error tolerance of 10^{-4}; the `expmimv` method uses 10,000 non-zeros, which may not achieve our desired tolerance, but identifies the right set with high probability, as evidenced in the experiment of Section 5.2.1.

Figure 5.4 compares our methods with the method `expmv` of [Al-Mohy and Higham, 2011] using the `single` and `half` accuracy settings (which we label as `expmv` and `half`, respectively). Our coordinate relaxation methods have highly variable runtimes, but can be very fast on graphs such as webbase (the point nearest 10^9 on the x-axis). We did not run the `gexpm` function for matrices larger than the livejournal graph.

5.2.3 Runtime scaling

The final experimental study we conduct attempts to better understand the runtime scaling of the `gexpmq` method. This method yields a prescribed accuracy ε more rapidly than `gexpm`, but the study

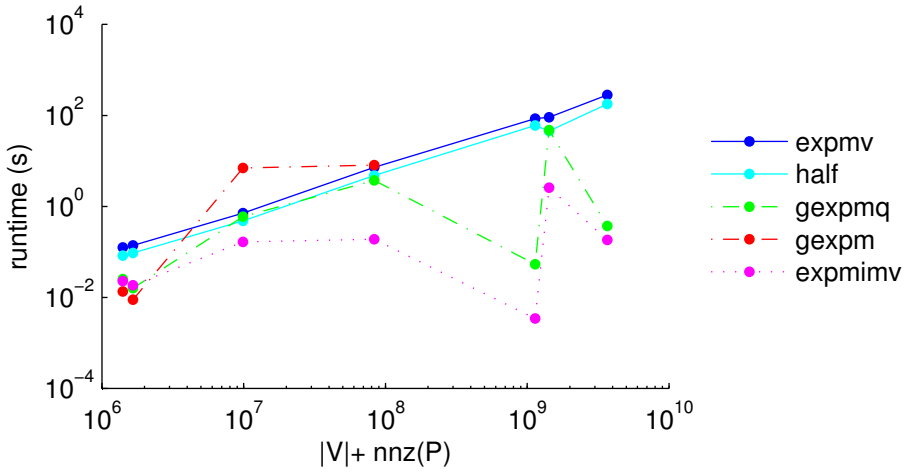

Figure 5.4. Runtime experiments on large real-world networks, comparing our matrix exponential algorithms with competing algorithms.

with real-world networks did not show a clear relationship between error and runtime. We conjecture that this is because the edge density varies too much between our graphs. Consequently, we study the runtime scaling on forest-fire synthetic graphs [Leskovec et al., 2007]. We use a symmetric variation on the forest-fire model with a single "burning" probability. We vary the number of vertices generated by the model to get graphs ranging from around 10,000 vertices to around 100,000,000 vertices.

Figure 5.5 shows the distribution of runtimes for the `gexpm` method on two forest-fire graphs (left: $p_f = 0.4$, middle, $p_f = 0.48$) of various graph sizes, where graph size is computed as the sum of the number of vertices and the number of non-zeros in the adjacency matrix. The thick line is the median runtime over 50 trials, and the shaded region shows the 25% to 75% quartiles (the shaded region is very tight for the second two figures). With $p_f = 0.48$, the graph is fairly dense – more like the friendster network – whereas the graph with $p_f = 0.4$ is highly sparse and is a good approximation for the webbase graph. Even with billions of edges, it takes less than 0.01 seconds for `gexpmq` to produce a solution with 1-norm error $\varepsilon = 10^{-4}$ on this sparse graph. For $p_f = 0.48$ the runtime grows with the graph size. The final plot (right) shows the relationship between the max-degree squared and the runtime in seconds. This figure shows that the runtime scales in a nearly linear relationship with the max-degree squared, as predicted by our theory. The large deviations from the line of best fit might be explained by the fact that only a single forest-fire graph was generated for each graph size.

We find that the scaling of d^2 seems to match the empirical scaling of the runtime (right plot), which is a plausible prediction based on Theorem 4.3.1. (Recall that one of the log factors in the bound of Theorem 4.3.1 arose from the heap updates in the `gexpm` method.) These results show that

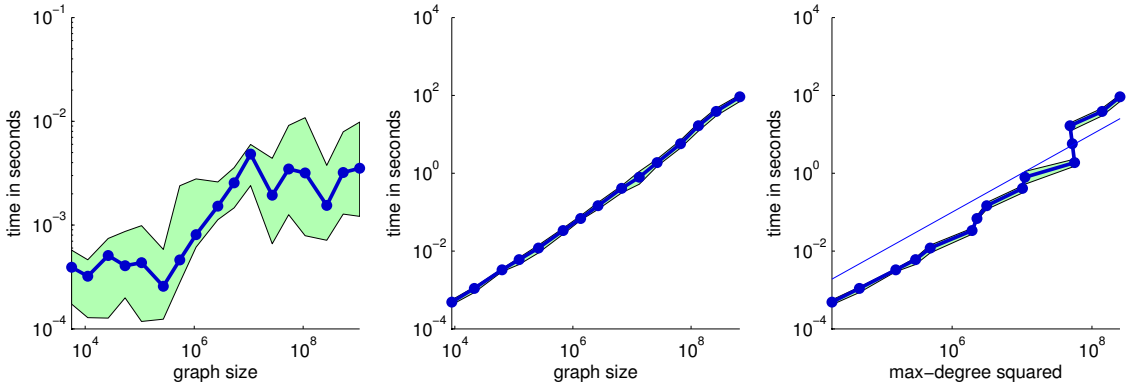

Figure 5.5. Runtime scaling of our matrix exponential algorithms on synthetic forest-fire graphs of different sizes.

our method is extremely fast when the graph is sufficiently sparse, but does slow down when running on networks with higher edge density.

6. OPTIMALLY ROUNDED MATRIX-VECTOR PRODUCTS

In this chapter we shift our focus from algorithms that are specifically designed for the matrix exponential to a broader theme that underlies all of our matrix exponential algorithms. The previous methods we presented can essentially be understood as using rounded (or "incomplete") matrix vector products to rapidly compute terms of a Taylor polynomial. Therefore, in this section our overarching goal is to speed up repeated matrix-vector products with a sparse matrix $\mathbf{A}$ by using strategic rounding to reduce fill-in. In this section we consider how to perform rounding on a *single* product $\hat{\mathbf{x}} \approx \mathbf{Ab}$ in a way that is optimal in terms of either the total flops performed or the total number of columns accessed (i.e. the number of non-zeros of $\mathbf{b}$ used).

Minimizing the number of either flops or memory accesses performed in computing a matrix vector product requires finding an optimal subset of entries of $\mathbf{b}$ to round to zero. It turns out that this is equivalent to the well-studied Knapsack Problem from combinatorial optimization, and we derived our results from that perspective. However, we give a self-contained presentation in the language of a matrix-vector product, rather than discuss the Knapsack Problem explicitly. The Knapsack Problem has a known greedy approximation algorithm that is guaranteed to find a solution within a factor of 2 of the optimal [Dantzig, 1957]. We propose a modified version of this greedy algorithm, and demonstrate that our method runs in $O(m)$ where m is the size of the Knapsack problem; this improves on the original greedy algorithm which runs in $Om \log m)$.

We organize the rest of the chapter as follows. First we describe the optimization problem for determining the optimal set of vector entries to round to zero (Section 6.1). Next we develop a data structure that enables fast approximate sorting (Section 6.2); this is the key to the speedup offered by our approximation algorithm. Then we present a novel algorithm for approximately solving the optimization problem; our algorithm improves on the standard greedy approximation algorithm for the Knapsack Problem (Section 6.3). Finally we prove that our algorithm (1) has runtime linear in the number of non-zeros of the vector being rounded, and (2) guarantees an approximate solution that reduces work by an amount within a constant factor of the optimal. The constant depends on a parameter, θ, of the algorithm which is discussed in more detail below, but the parameter value we use in practice makes the constant factor equal to 4, i.e. our algorithm avoids at least 1/4th of the maximum possible amount of work that could be avoided. In other words, our algorithm will round entries in the vector $\mathbf{b}$ such that we round to zero at least 1/4 of the maximum possible amount of entries that can be rounded to zero while still producing an approximation of the prescribed accuracy.

6.1 Rounding as an optimization problem

We want to approximate $\mathbf{x} = \mathbf{Ab}$ by first rounding entries of $\mathbf{b}$ to zero to decrease the amount of work required to perform an approximation of the product $\mathbf{Ab}$. At the same time, we do not want to round too many entries of $\mathbf{b}$ to zero or else we will accrue too much error. Let $\hat{\mathbf{b}}$ denote an approximation to the vector $\mathbf{b}$ that we obtain by rounding some entries of $\mathbf{b}$ to zero. Then we want to determine the set S of entries of $\mathbf{b}$ to round to zero such that we minimize the amount of work required to compute $\mathbf{A}\hat{\mathbf{b}}$ subject to the constraint that $\|\mathbf{b} - \hat{\mathbf{b}}\| \leq \varepsilon$ for some prescribed error ε.

To formalize this, we fix some notation. We define a weight vector $\mathbf{w} = |\mathbf{b}|$, i.e. $\mathbf{w}$ is the entry-wise absolute value of the vector $\mathbf{b}$. Let $\text{nnz}(\mathbf{b})$ be the number of nonzero entries in the vector $\mathbf{b}$, and denote by $\mathbf{A}(:, j)$ column j of the matrix $\mathbf{A}$. Let $\mathbf{e}$ be the vector of all 1s of appropriate dimension. For any set T and any vector $\mathbf{v}$ the vector $\mathbf{v}_T$ denotes the vector $\mathbf{v}$ with entries outside T set to zero. If the vector $\hat{\mathbf{b}}$ is the approximation to $\mathbf{b}$ obtained from rounding to zero the set of entries S in the vector $\mathbf{b}$, then we can write $\hat{\mathbf{b}} = (\mathbf{b} - \mathbf{b}_S)$. The 1-norm error of this approximation is $\|\mathbf{b} - (\mathbf{b} - \mathbf{b}_S)\|_1$, and we can express it as $\|\mathbf{b} - (\mathbf{b} - \mathbf{b}_S)\|_1 = \|\mathbf{b}_S\|_1 = \mathbf{w}^T\mathbf{e}_S$. Note that this analysis could be carried out with other sub-multiplicative norms, but the analysis is cleanest with the 1-norm because of the relationship $\|\mathbf{b}_S\|_1 = \mathbf{w}^T\mathbf{e}_S$.

Next we want to formally express the amount of work performed in computing the approximate product $\mathbf{A}\hat{\mathbf{b}}$. We encode this amount of work as a *cost* vector, $\mathbf{c}$, for which we discuss specific instances below. To compute the sparse matrix-vector product $\mathbf{A}\hat{\mathbf{b}}$, for each nonzero entry of $\hat{\mathbf{b}}$ we must access the corresponding column of the matrix $\mathbf{A}$. We can express this formally by writing

$$\mathbf{A}\hat{\mathbf{b}} = \sum_{j=1}^{n} \mathbf{A}(:, j)\hat{b}_j$$

and note that for each entry $\hat{b_j}$ that is nonzero we must perform $O(\text{nnz}(\mathbf{A}(:, j)))$ work in computing the term $\mathbf{A}(:, j)\hat{b}_j$. The total amount of work performed in computing $\mathbf{A}\hat{\mathbf{b}}$ can then be expressed as follows. We define a cost vector $\mathbf{c}$ by $c_j = \text{nnz}(\mathbf{A}(:, j))$. If $\hat{\mathbf{b}} = \mathbf{b}_T$, for some set of entries T, then we can express the cost of computing $\mathbf{A}\hat{\mathbf{b}}$ to be the quantity $\mathbf{c}^T\mathbf{e}_T$. So we have that the total cost of performing the exact product $\mathbf{Ab}$ is $\mathbf{c}^T\mathbf{e}$, and the cost of performing the product after rounding to zero the entries in the set S is then $\mathbf{c}^T(\mathbf{e} - \mathbf{e}_S)$. The general optimization problem is then to choose S to minimize $\mathbf{c}^T(\mathbf{e} - \mathbf{e}_S)$ subject to the constraint $\mathbf{w}^T\mathbf{e}_S \leq \varepsilon$. However, minimizing $\mathbf{c}^T(\mathbf{e} - \mathbf{e}_S)$ is equivalent to maximizing the simpler expression $\mathbf{c}^T\mathbf{e}_S$, and so we focus on the optimization problem

$$\begin{array}{ll} \underset{S}{\text{maximize}} & \mathbf{c}^T\mathbf{e}_S \\ \text{subject to} & \mathbf{w}^T\mathbf{e}_S \leq \varepsilon \end{array}, \tag{6.1}$$

which we remark is an integer-valued optimization problem. If the cost we want to optimize is the total number of flops performed in computing $\mathbf{Ab}$, then the cost vector $\mathbf{c}$ should have entry j equal

to the number of nonzero entries in column j of $\mathbf{A}$, i.e. $\mathbf{c}_j = \text{nnz}(\mathbf{A}(:, j))$. If instead we are concerned with optimizing the number of columns we access, then the cost of each entry b_j is 1, as each entry corresponds to one column. Hence, we set $\mathbf{c} = \mathbf{e}$ in this case. Note that in the case that every column of $\mathbf{A}$ has exactly the same number of nonzero entries, these two different cost functions lead to equivalent optimization problems. This situation arises, for example, in the case that $\mathbf{A}$ is the adjacency matrix or Laplacian matrix of a degree-regular graph.

This optimization problem (with either cost vector) can be recast in terms of the Knapsack Problem. In the next section we present our greedy knapsack algorithm that approximately solves the Knapsack Problem (and, hence, the optimization problem (6.1)) within a factor of 2θ of the optimal solution, where $\theta > 1$ is a tunable parameter that we discuss in detail below.

We remark that the simplicity of the second version of the problem in (6.1) (e.g. having the cost values all be 1) enables a known, straight-forward, and efficient solution: First sort the vector of weights, $\mathbf{w}$, and then take S to be the largest set of weights possible. In the context of rounding a matrix-vector product, this corresponds to rounding to zero the smallest entries until the tolerance ε is reached. This maximizes the number of column accesses avoided by rounding.

6.2 A faster greedy knapsack algorithm, and near-optimal rounding

Our approximate method for the Knapsack Problem relies on a data structure that enables constant-time operations for insertion and for retrieval of near-min entries. This is in contrast with, for example, a min-heap data structure which would allow constant-time retrieval of the exact-minimum, but at the cost of requiring $\log n$ time insertions. The data structure we present enables us to remove this $\log n$ factor to obtain a linear time approximation algorithm. We present this data structure here before describing our greedy Knapsack algorithm itself.

Shelf structure

The motivation of the shelf structure is to store entries such that we have fast access to small entries, without requiring costly update times (as is the case with a heap). We do this by storing entries of different orders of magnitude in different arrays, such that all entries in a single array are within a constant factor of each other. Any constant $\theta > 1$ can be used, but in practice we use $\theta = 2$ because this enables faster computations of log, which is an essential subroutine for the fast insertion operation.

A single array in the shelf holds all entries in the whole structure that have value r satisfying

$$2^k \leq r < 2^{k+1}$$

for values of $k \geq 0$. We call these arrays "shelves". This particular shelf we denote by H_k. Note that, before inserting the entries into the shelf, we can normalize the set of entries by taking absolute values and dividing each entry by the smallest value in the set. This guarantees every entry in the shelf structure has absolute value ≥ 1, and so all entries r will satisfy $2^k \leq r < 2^{k+1}$ for some $k \geq 0$. This is helpful because it guarantees that the smallest magnitude element will always rest in shelf H_0—the "top" shelf. This makes it easier to implement the shelf system. (The smallest element can be determined prior to inserting entries into the shelf by a single scan over all entries. Because this is a linear time operation performed just once, it does not affect the overall linear runtime.)

Entry insertion Let r_{MIN} be the minimum value to be placed in the shelf. To determine the shelf, k, where a value r should be inserted, compute

$$k = \lceil \log_2(r/r_{\text{MIN}}) \rceil.$$

This can be done efficiently by checking the exponent of the floating point representation of r/r_{MIN}. Since checking the exponent returns the appropriate integer, determining k requires just one division and one exponent check.

Entry retrieval To retrieve an element within a factor of 2 of the minimum, take any element from the top-most non-empty shelf, H_k. Determining the top-most non-empty shelf requires work bounded by the number of shelves. This number is $\lceil \log(r_{\text{MAX}}/r_{\text{MIN}}) \rceil$ and in practice it is a small constant. However, theoretically the number is unbounded unless the vector $\mathbf{b}$ has bounds on its entries.

Top shelf If our set of values is such that $r_{\text{MAX}}/r_{\text{MIN}}$ is arbitrarily large, then filtering the entries into the shelf system would require an arbitrarily large number of shelves. For example, the set of values $\{2^{2^0}, 2^{2^1}, \ldots, 2^{2^n}\}$ would require a shelf system with $\Theta(2^n)$ different shelves. However, in our context there are bounds on the values in question that prevent the number of shelves from growing too large.

6.3 Near-optimal rounding algorithm

With the details of the shelf structure laid down, now we describe how to use that data structure for the Knapsack Problem. We want to solve the optimization problem in (6.1),

$$\begin{array}{ll} \underset{S}{\text{maximize}} & \mathbf{c}^T \mathbf{e}_S \\ \text{subject to} & \mathbf{w}^T \mathbf{e}_S \leq \varepsilon \end{array}.$$

Recall that the original goal here is that we want to approximate the matrix-vector product $\mathbf{Ab}$; in order to minimize the amount of work we perform in computing this product, we first want to determine a set S of entries in $\mathbf{b}$ that we will round to zero so as to minimize the cost of computing $\mathbf{A}\hat{\mathbf{b}}$.

To proceed, first compute the ratios $r_j = w_j/c_j$ for all nonzero entries of $\mathbf{w}$, and insert each ratio r_j into the shelf structure. We can assume no entry has zero cost, i.e. we assume $c_j \neq 0$ for all j; if an entry has $c_j = 0$ then in the context of the original problem (computing $\mathbf{Ab}$) no work is incurred by including entry j in the computation $\mathbf{Ab}$, and so we ignore entry j in this optimization problem.

Next, use the shelf system to repeatedly retrieve near-min entries and add the corresponding entry to the solution set S, keeping a running sum of the total weight in S. Relabel the entries so that the values w_j, r_j, and c_j correspond to the jth entry retrieved from the shelf. When the total weight exceeds the tolerance, i.e. $w_1 + \cdots + w_p > \varepsilon$, we call the index p the *break index*. The algorithm then returns as the solution set, S, either the set $\{1, \ldots, p-1\}$, or the break index alone, $\{p\}$, whichever set has greater total cost. Recall that this set, S, is the set of indices of the vector $\mathbf{b}$ which we round to zero before performing the matrix-vector product $\mathbf{Ab}$.

This algorithm requires $\Theta(\text{nnz}(\mathbf{w}))$ insertions and retrievals, and a small, constant number of other operations. Recall that a 0 in $\mathbf{w}$ corresponds to a 0 in the original vector $\mathbf{b}$. Assuming that the shelf insertion and retrieval subroutines are constant time, then, this proves the algorithm runs in time $\Theta(\text{nnz}(\mathbf{b}))$. Next, we present brief experimental results before we formally state bounds on the runtime and accuracy of the algorithm.

Experimental results We present a small experiment here to illustrate the potential utility of knapsack rounded matrix-vector products. We study the number of column accesses used to compute a power of a graph matrix times a sparse vector. In particular, we count the number of column accesses used to perform the computation exactly, and compare this with the number of column accesses used if we perform knapsack rounding between successive matrix-vector products.

More specifically, one trial of the experiment consists of the following. Select a node j uniformly at random from the graph, and compute $\mathbf{x} = \mathbf{P}^6\mathbf{e}_j$ exactly, as well as using our implementation of the knapsack-rounded matrix-vector product procedure. The rounded matrix-vector products use an error tolerance of $\varepsilon = 10^{-2}$. Here $\mathbf{P}$ is the random-walk transition matrix for a version of the webgraph, `webbase-2001` [Hirai et al., 2000, Boldi and Vigna, 2005], that has $118,142,155$ nodes and $992,844,891$ directed edges.

We perform the experiment 50 times, selecting a new node j each time, uniformly at random from the graph, with replacement. If the node has degree less than 5, a new node is chosen (this is to avoid selecting dangling nodes). The result is that the knapsack procedure uses, on average, uses

only 39% of the number of column accesses used in the exact computation, and never produces an error larger than $\|\mathbf{x} - \hat{\mathbf{x}}\|_1 \leq 0.05$.

Theorem 6.3.1 *Let $\boldsymbol{b}$ be a vector, $\boldsymbol{A}$ be a matrix we intend to multiply by $\boldsymbol{b}$, and let $\boldsymbol{c}$ be a cost vector such that c_j represents the cost associated with accessing column j of the matrix $\boldsymbol{A}$. Define efficiency ratios $r_j = |b_j|/c_j$, and denote the maximum and minimum ratios to be r_{MAX} and r_{MIN}. Given an error tolerance ε, the approximate knapsack algorithm presented in Section 6.2 determines a subset S of the entries of $\boldsymbol{b}$ that can be rounded to zero such that the resulting approximation $\hat{\boldsymbol{b}} = \boldsymbol{b} - \boldsymbol{b}_S$ satisfies $\|\boldsymbol{b} - \hat{\boldsymbol{b}}\|_1 \leq \varepsilon$ and such that the amount of matrix-vector product work that is avoided, $\boldsymbol{c}_S$, is within a factor of 2θ of the optimal, where θ is the parameter used in the shelf structure. The procedure requires $\Theta(nnz(\boldsymbol{b}) + \log(r_{MAX}/r_{MIN}))$ work.*

If the ratios satisfy $\log(r_{\text{MAX}}/r_{\text{MIN}}) \leq \text{nnz}(\mathbf{b})$ then the procedure runs in time $\Theta(\text{nnz}(\mathbf{b}))$. This is linear in the number of items (entries of $\mathbf{b}$), which improves on the standard $O(n \log n)$ greedy algorithm for attaining a 1/2-approximate solution to the knapsack problem. We remark that we phrase Theorem 6.3.1 in the language of rounding to zero some entries of the vector $\mathbf{b}$; this corresponds to choosing items to put into the knapsack such that cost is maximized without violating the weight constraint of the knapsack.

Proof The runtime bound of $\Theta(\text{nnz}(\mathbf{b}) + \log(r_{\text{MAX}}/r_{\text{MIN}}))$ was already proved in Section 6.2. The accuracy proof is similar to a standard proof that shows a well-known greedy knapsack algorithm produces a 2-approximation.

Let $s = \text{nnz}(\mathbf{b})$. Then in the notation of Section 6.3, let the sorted set of ratios of weight and cost for the s non-zeros be

$$r_1 = \tfrac{w_1}{c_1} \leq r_2 \leq \cdots \leq r_s = \tfrac{w_s}{c_s}.$$

Consider the following convex relaxation of the integer-valued optimization problem in (6.1). Instead of searching for a solution $\mathbf{e}_S$ in the space of $\{0,1\}$-valued vectors, here we maximize over the set of vectors $\mathbf{s} \in [0,1]^s$, i.e.

$$\begin{array}{ll} \underset{\mathbf{s}\in[0,1]^s}{\text{maximize}} & \mathbf{c}^T\mathbf{s} \\ \text{subject to} & \mathbf{w}^T\mathbf{s} \leq \varepsilon \end{array}. \tag{6.2}$$

Just as problem (6.1) is equivalent to a Knapsack Problem, the formulation in (6.2) is equivalent to the so-called Fractional Knapsack Problem. This formulation is a linear program, and its solution has the following known form [Dantzig, 1957].

First, use the ordering on the sorted ratios to determine the *break index*, the index k such that $\sum_{j=1}^{k-1} w_j \leq \varepsilon < \sum_{j=1}^{k} w_j$. The standard greedy algorithm returns as the solution set S either the first $k-1$ indices, or the break index alone, whichever has greater cost. In this fractional

setting, we can include the first $k-1$ items in their entirety, then include the break index in a fractional amount, $\alpha \in [0,1]$, so as not to exceed the weight constraint. The fraction is equal to $\alpha = (\varepsilon - \sum_{j=1}^{k-1} w_j)/w_k$, so that $\alpha w_k + \sum_{j-1}^{k-1} w_j = \varepsilon$ holds exactly. Then the optimal solution to Equation (6.2) is $\mathbf{s}^T = [1, \ldots, 1, \alpha, 0, \ldots, 0]$, where the α is in entry k of $\mathbf{s}$. Denote the optimal value of $\mathbf{c}^T\mathbf{s}$ by $\hat{V}_{\text{OPT}} = \alpha c_k + \sum_{j=1}^{k-1} c_j$.

Next we show that the value obtained by our approximation algorithm, V, is within a factor of 2θ of $\hat{V}_{\text{OPT}}$. Because Equation (6.2) is a relaxation of the original optimization problem, we know that the optimal value of (6.2), $\hat{V}_{\text{OPT}}$, must be greater than the optimal value V_{OPT} of the original problem, (6.1). Hence, proving V is within a factor of 2θ of $\hat{V}_{\text{OPT}}$ implies that our approximate solution is within a factor of 2θ of the optimal V_{OPT}.

The true solution to (6.2) requires exactly sorting the ratios, but we have access to only a partial sorting. More precisely, we cannot select the entry with the smallest ratio, but we can select an entry that has a ratio within a factor of θ of the smallest. We draw near-min entries from the shelf system until we reach a break index, m. Let d_j be the ratio of the jth item selected from the shelf, and let the corresponding item's weight and cost be denoted by $\tilde{w}_j$ and $\tilde{c}_j$, respectively.

Claim: After we've selected these m items, we claim that we can assume that the ratios of the items we've selected satisfy $d_j \leq \theta r_1$ for all j, without losing any generality. To prove this, note that if none of the items we select happens to be the item corresponding to r_1, then this claim is satisfied. This is because the shelf guarantees that we always select an item with a ratio within a factor of θ of the smallest ratio *still present in the shelf*. If r_1 is never selected, it is never removed from the shelf, and so all ratios selected would satisfy $d_j \leq \theta r_1$.

To complete the proof of this claim, we now compare the list of items that the exact-sort algorithm selects and the list of items that the shelf-sort algorithm selects. Because we want to prove a relationship of the shelf-approximation's value to the value of the greedy-approximation, we can look at the values of just the items where the two algorithms differ. That is, we can "remove" from each list any item that appears in both lists. We call such items *shared* items, and the items that remain after their removal, *non-shared* items.

To proceed, relabel the ratios of the items in the exact-sort list so that $r_1 \leq r_2 \leq \cdots \leq r_t$ are the sorted ratios corresponding to the non-shared items. Similarly, relabel the shelf-algorithm's ratios $d_1, d_2, \ldots, d_p$ for non-shared items. Finally, observe that each non-shared item selected by the shelf-algorithm must have a ratio within a factor of θ of r_1—this is because r_1 is the smallest ratio present in the shelf system. Thus, for each j we have $d_j \leq \theta r_1$.

Now we can lower-bound the value attained in (6.2) by the shelf-approximation as follows. First, by removing the shared items from the item-list of both algorithms, we have effectively altered the

weight constraint. The new weight limit, ε', is a limit on the total weight of the non-shared items. Then the two sets have weights that satisfy:

$$\tilde{\alpha}\tilde{w}_p + \sum_{j=1}^{p-1} \tilde{w}_j \quad = \quad \varepsilon' \quad = \quad \alpha w_t + \sum_{j=1}^{t-1} w_j,$$

where p is the break index for the shelf-sort algorithm, and t is the break index of the exact-sort algorithm. Similarly, the cost values achieved by the two algorithms on the set of *shared* items is identical, so we consider only the total cost value obtained on the non-shared items. Denote by V and by V_{OPT} the total values of the non-shared items of the shelf-approximation and the optimal solution, respectively. By definition we have that $V = \tilde{\alpha}\tilde{c}_p + \sum_{j=1}^{p-1} \tilde{c}_j$ and $\hat{V}_{\text{OPT}} = \alpha c_t + \sum_{j=1}^{t-1} c_j$. Next we show that $2\theta V \geq \hat{V}_{\text{OPT}}$.

First note that in the expression for V we can express the values $\tilde{c}_j$ as $\tilde{w}_j/d_j$ (since the ratios are defined by $d_j = \tilde{w}_j/\tilde{c}_j$): $V = \tilde{\alpha}\tilde{w}_p/d_p + \sum_{j=1}^{p-1} \tilde{w}_j/d_j$. Using the fact proved above that $d_j \leq \theta r_1$, we can lowerbound $1/d_j$ with $1/(\theta r_1)$. Substituting yields $V \geq \tilde{\alpha}\tilde{w}_p/(\theta r_1) + \sum_{j=1}^{p-1} \tilde{w}_j/(\theta r_1)$. Some algebra allows us to write $V \geq \left(\tilde{\alpha}\tilde{w}_p + \sum_{j=1}^{p-1} \tilde{w}_j\right)/(\theta r_1)$. The numerator here is one expression for ε', so we have $V \geq \varepsilon'/(\theta r_1)$. Expressing ε' with the weights $w_1, \ldots, w_t$ yields $V \geq \left(\alpha w_t + \sum_{j=1}^{t-1} w_j\right)/(\theta r_1)$. Further algebra yields

$$\begin{aligned}
\theta V &\geq \alpha w_t/r_1 + \sum_{j=1}^{t-1} w_j/r_1 \\
&\geq \alpha w_t/r_t + \sum_{j=1}^{t-1} w_j/r_j && \text{because } r_1 \leq r_j \\
&= \alpha c_t + \sum_{j=1}^{t-1} c_j, && \text{using } r_j = w_j/c_j
\end{aligned}$$

which is $\hat{V}_{\text{OPT}}$. This proves $\theta V \geq \hat{V}_{\text{OPT}}$. Next we use the relationship just established to prove the theorem.

Set $V_\theta := \max\{\tilde{c}_p, \sum_{j=1}^{p-1} \tilde{c}_j\}$ and note that this is the cost achieved by our algorithm. Observe that $2V_\theta \geq \tilde{c}_p + \sum_{j=1}^{p-1} \tilde{c}_j$, and since the scalar $\tilde{\alpha}$ is between 0 and 1 we then have $2V_\theta \geq \tilde{\alpha}\tilde{c}_p + \sum_{j=1}^{p-1} \tilde{c}_j = V$. But we proved above that $\theta V \geq \hat{V}_{\text{OPT}}$, and so we have $2\theta V_\theta \geq \hat{V}_{\text{OPT}}$. By our remark above, we know $\hat{V}_{\text{OPT}} \geq V_{\text{OPT}}$, and so we have proved that the cost value attained, V_θ, satisfies $2\theta V_\theta \geq V_{\text{OPT}}$, as desired. ■

In this section we considered the problem of performing a single matrix-vector product with optimal rounding. This result is one step toward a broader goal of applying sparsifying procedures within repeated matrix-vector products so as to minimize fill-in during the evaluation of a vector times a general polynomial of a matrix. This motivation goes far beyond one of the main concerns of this thesis, namely approximating a vector times the matrix exponential: fill-in minimizing matrix-vector

products for a vector times a polynomial of a matrix could have applications to computing general functions of matrices, and even solving linear systems and eigensystems, as all of these problems use the matrix vector product as a fundamental unit of computation. We leave as future work the application of our new method to these other computational problems.

7. DIFFUSIONS FOR LOCAL GRAPH ANALYSIS

In previous sections we considered computing columns of matrix functions, $f(\mathbf{P})\mathbf{e}_j$, to a prescribed 1-norm accuracy, and studying localization behavior in a strong sense, i.e. the sparsity of an approximation attaining a specified 1-norm accuracy. Here we shift our attention to *graph diffusions* (matrix functions with a type of normalization discussed below), motivated by a set of applications for which accuracy is better measured using a degree-normalized infinity-norm. Applications include tasks like link prediction [Kunegis and Lommatzsch, 2009], community detection [Chung, 2007a], and node ranking [Estrada, 2000, Farahat et al., 2006, Estrada and Higham, 2010], and for a more comprehensive list of PageRank applications see my adivsor's recent survey [Gleich, 2015a]. For such applications, rather than computing the entire length n diffusion vector, it often suffices simply to identify a small set of nodes that are most important to the target seed node(s).

A graph diffusion is any sum of the following form:

$$\mathbf{f} = \sum_{k=0}^{\infty} c_k \mathbf{P}^k \mathbf{s}$$

where $\sum_k c_k = 1$ and $\mathbf{s}$ is a stochastic vector (that is, it is non-negative and sums to one). Note that this is simply a matrix function, the function is $f(x) = \sum_{k=0}^{\infty} c_k x^k$. However, the power series perspective on a function of a matrix emphasizes the following understanding of diffusions. Intuitively, a diffusion captures how a quantity of s_i material on node i *flows* through the graph. The terms c_k provide a decaying weight that ensures that the diffusion eventually dissipates. The kth power of the random walk transition matrix, $\mathbf{P}^k$, gives the distribution of the diffusion after k steps of a random walk, and so c_k can be understood as weighting walks of length k in the diffusion. We are interested in the diffusions of single nodes or neighborhood sets of a single vertex; and so in these cases $\mathbf{s} = \mathbf{e}_i$ or $\mathbf{s} = \sum_{i \in S} \mathbf{e}_i / |S|$ for a small set S. We call the origins of the diffusion the *seeds*.

Given an estimate of a diffusion $\mathbf{f}$ from a seed, one can try to use the vector to obtain a community. Certain diffusions like PageRank and the heat kernel have been proved to yield good conductance sets via a sweep-cut procedure over the approximate diffusion vector [Andersen et al., 2006a, Chung, 2007b]. The sweep-cut procedure involves computing $\mathbf{D}^{-1}\mathbf{f}$, sorting the nodes in descending order by their magnitude in this vector, and computing the conductance of each prefix of the sorted list. Due to the properties of conductance, there is an efficient means of computing all of these conductances. One would then return the set of smallest conductance as the community around the seeds.

The personalized PageRank diffusion. One of the most well-known instances of this framework is the personalized PageRank diffusion. Fix $\alpha \in (0, 1)$. Then $\mathbf{p}$ is defined:

$$\mathbf{p} = (1 - \alpha) \sum_{k=0}^{\infty} \alpha^k \mathbf{P}^k \mathbf{s}. \tag{7.1}$$

The properties of this diffusion have been studied extensively. In particular, Andersen et al. [Andersen et al., 2006a] establish a local Cheeger inequality using a particular algorithm called "push" that locally distributes mass. The local Cheeger inequality informally states that, *if* the seed is nearby a set with small conductance, *then* the result of the sweep procedure is a set with a related conductance. Moreover, they show that their "push" algorithm estimates $\mathbf{f}$ with an error ε in a degree-weighted norm by looking at $\frac{1}{(1-\alpha)\varepsilon}$ edges.

The heat kernel diffusion. Another instance of the same framework is the heat kernel diffusion [Chung, 2007a, 2009]. It simply replaces the weights c_k with $e^{-t}t^k/k!$:

$$\mathbf{h} = \left(\sum_{k=0}^{\infty} e^{-t} \frac{t^k}{k!} \mathbf{P}^k \right) \mathbf{s} = \exp\left\{-t(\mathbf{I} - \mathbf{P})\right\} \mathbf{s}. \tag{7.2}$$

While it was known that estimating $\mathbf{h}$ gave rise to a similar type of local Cheeger inequality [Chung, 2009]; until Chung and Simpson's Monte Carlo approach [Chung and Simpson, 2013] and our own deterministic algorithm [Kloster and Gleich, 2014], no methods were known to estimate this quantity efficiently.

Many recent studies have focused on the computation, behavior, and applications of these and similar diffusions. Individual random walk vectors have been proposed for identifying cuts of good conductance [Andersen and Lang, 2006]. The first efficient algorithm has surfaced for computing the so-called *time-dependent PageRank* diffusion, which in a sense "interpolates" between PageRank and heat kernel, [Avron and Horesh, 2015]. Recent work on a *local spectral* clustering method implicitly computes a diffusion by computing terms of the random walk, $\mathbf{P}^k\mathbf{s}$, then solving an optimization problem over the subspace spanned by these terms [Li et al., 2015]. The different diffusions' coefficients can be thought of as placing different weights on walks of different lengths in a random walk process. Figure 7.1 shows a comparison of how random walks of different lengths are weighted by the PageRank (dashed blue curve), heat kernel (sold red curve), and time-dependent PageRank (green dot-dash curve) diffusions. Each curve in the figure represents the coefficients of $(\mathbf{A}\mathbf{D}^{-1})^k$ in a sum of walks. The dotted blue lines give α^k, and the red give $t^k/k!$, for the indicated values of α and t. The time-dependent PageRank curve shown uses the parameters $\gamma = 5.0$ and $\alpha = 0.85$.

As more exotic types of diffusions are used for local clustering, we would like constant-time algorithms for approximating such diffusions similar to the widely-used efficient "pprpush" algorithm for PageRank [Andersen et al., 2006a]. In this chapter we describe the first deterministic algorithm

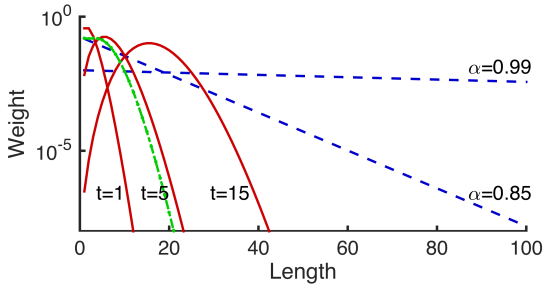

Table 7.1.
Path weights for different diffusions.

for the heat kernel diffusion (Section 7.1), then generalize our work to apply to a broader class of diffusions (Section 7.4). We demonstrate that both of our algorithmic frameworks can compute desired diffusions in *constant time* (Theorems 7.2.1 and 7.4.1). We emphasize that, in contrast with other chapters of this thesis, our results in this chapter apply specifically to diffusions on *undirected graphs*.

As the first deterministic algorithm for computing the heat kernel, our new algorithm is more suitable than existing randomized methods for comparing the properties of the heat kernel with the PageRank diffusion. We present experiments showing that the heat kernel performs comparably with PageRank in terms of speed and conductance, and outperforms PageRank in a ground-truth community detection experiment on real-world networks (Section 7.3).

7.1 Heat kernel diffusions in constant time

The overall idea of the local clustering algorithm is to approximate a heat kernel vector of the form

$$\mathbf{h} = \exp\{-t(\mathbf{I} - \mathbf{P})\}\,\mathbf{s}$$

so that we can perform a sweep over $\mathbf{h}$. Here we describe a coordinate-relaxation method, which we call `hk-relax`, for approximating $\mathbf{h}$. This algorithm is rooted in the work presented in Sections 4.1 and 5.1 on computing an accurate column of $\exp\{\mathbf{P}\}$; but is heavily tuned to the objective described below. Thus, while the overall strategy is classical – just as the PageRank push method is a classic relaxation method – the simplifications and efficient implementation are entirely novel. In particular, the new objective in this section enables us to get a constant runtime bound independent of any property of the graph, which differs markedly from the algorithms we presented in Chapters 4 and 5.

Our objective. Recall that the final step of finding a small conductance community involves dividing by the degree of each node. Thus, our goal is to compute $\mathbf{x} \approx \mathbf{h}$ satisfying the degree weighted bound:

$$\|\mathbf{D}^{-1} \exp\{-t(\mathbf{I} - \mathbf{P})\}\, \mathbf{s} - \mathbf{D}^{-1}\mathbf{x}\|_\infty < \varepsilon.$$

By using standard properties of the matrix exponential, we can factor $\exp\{-t(\mathbf{I} - \mathbf{P})\} = e^{-t} \exp\{t\mathbf{P}\}$ and scale by e^t so that the above problem is equivalent to computing $\mathbf{y}$ satisfying $\|\mathbf{D}^{-1}(\exp\{t\mathbf{P}\}\, \mathbf{s} - \mathbf{y})\|_\infty < e^t\varepsilon$. The element-wise characterization is that $\mathbf{y}$ must satisfy:

$$|e^t\mathbf{h}_i - \mathbf{y}_i| < e^t\varepsilon d_i \tag{7.3}$$

for all i. A similar weighted objective was used in the push algorithm for PageRank [Andersen et al., 2006a].

Outline of algorithm. To accomplish this, we first approximate $\exp\{t\mathbf{P}\}$ with its degree N Taylor polynomial, $T_N(t\mathbf{P})$, and then we compute $T_N(t\mathbf{P})\mathbf{s}$. But we use a large, implicitly formed linear system to avoid explicitly evaluating the Taylor polynomial. Once we have the linear system, we state a relaxation method in the spirit of Gauss-Seidel and the PageRank push algorithm in order to compute an accurate approximation of $\mathbf{h}$.

7.1.1 Taylor Polynomial for exp{X}

Determining the exponential of a matrix is a sensitive computation with a rich history [Moler and Van Loan, 1978, 2003]. For a general matrix $\mathbf{G}$, an approximation via the Taylor polynomial,

$$\exp\{\mathbf{G}\} = \sum_{k=0}^{\infty} \tfrac{1}{k!}\mathbf{G}^k \approx \sum_{k=0}^{N} \tfrac{1}{k!}\mathbf{G}^k,$$

can be inaccurate when $\|\mathbf{G}\|$ is large and $\mathbf{G}$ has mixed signs, as large powers $\mathbf{G}^k$ can contain large, oppositely signed numbers that cancel properly only in exact arithmetic. However, we intend to compute $\exp\{t\mathbf{P}\}\, \mathbf{s}$, where $\mathbf{P}$, t, and $\mathbf{s}$ are nonnegative, so the calculation does not rely on any delicate cancellations. Furthermore, our approximation need not be highly precise. We therefore use the polynomial $\exp\{t\mathbf{P}\}\, \mathbf{s} \approx T_N(t\mathbf{P}) = \sum_{k=0}^{N} \frac{t^k}{k!}\mathbf{P}^k$ for our approximation. For details on choosing N, see Section 7.1.5. For now, assume that we have chosen N such that

$$\|\mathbf{D}^{-1} \exp\{t\mathbf{P}\}\, \mathbf{s} - \mathbf{D}^{-1}T_N(t\mathbf{P})\mathbf{s}\|_\infty < \varepsilon/2. \tag{7.4}$$

This way, if we compute $\mathbf{y} \approx T_N(t\mathbf{P})\mathbf{s}$ satisfying

$$\|\mathbf{D}^{-1}T_N(t\mathbf{P})\mathbf{s} - \mathbf{D}^{-1}\mathbf{y}\|_\infty < \varepsilon/2,$$

then by the triangle inequality we will have

$$\|\mathbf{D}^{-1}\exp\{t\mathbf{P}\}\,\mathbf{s}-\mathbf{D}^{-1}\mathbf{y}\|_\infty<\varepsilon, \tag{7.5}$$

our objective.

7.1.2 Error weights

Using a degree N Taylor polynomial, `hk-relax` ultimately approximates $\mathbf{h}$ by approximating each term in the sum of the polynomial times the vector $\mathbf{s}$:

$$\mathbf{s}+\tfrac{t}{1}\mathbf{P}\mathbf{s}+\cdots+\tfrac{t^N}{N!}\mathbf{P}^N\mathbf{s}.$$

The total error of our computed solution is then a weighted sum of the errors at each individual term, $\frac{t^k}{k!}\mathbf{P}^k\mathbf{s}$. We show in Lemma 7.2.1 that these weights are given by the polynomials $\psi_k(t)$, which we define now. For a fixed degree N Taylor polynomial of the exponential, $T_N=\sum_{k=0}^{N}\frac{t^k}{k!}$, we define

$$\psi_k:=\sum_{m=0}^{N-k}\tfrac{k!}{(m+k)!}t^m \text{ for } k=0,\cdots,N. \tag{7.6}$$

These polynomials $\psi_k(t)$ are closely related to the ϕ functions central to exponential integrators in ODEs [Minchev and Wright, 2005]. Note that $\psi_0=T_N$.

To guarantee the total error satisfies the criterion (7.3) then, it is enough to show that the error at each Taylor term satisfies an ∞-norm inequality analogous to (7.3). This is discussed in more detail in Section 7.2.

7.1.3 Deriving a linear system

To define the basic step of the `hk-relax` algorithm and to show how the ψ_k influence the total error, we rearrange the Taylor polynomial computation into a linear system.

Denote by $\mathbf{v}_k$ the k^{th} term of the vector sum $T_N(t\mathbf{P})\mathbf{s}$:

$$T_N(t\mathbf{P})\mathbf{s}=\mathbf{s}+\tfrac{t}{1}\mathbf{P}\mathbf{s}+\cdots+\tfrac{t^N}{N!}\mathbf{P}^N\mathbf{s} \tag{7.7}$$

$$=\mathbf{v}_0+\mathbf{v}_1+\cdots+\mathbf{v}_N. \tag{7.8}$$

Note that $\mathbf{v}_{k+1} = \frac{t^{k+1}}{(k+1)!}\mathbf{P}^{k+1} = \frac{t}{(k+1)}\mathbf{P}\mathbf{v}_k$. This identity implies that the terms $\mathbf{v}_k$ exactly satisfy the linear system

$$\begin{bmatrix} \mathbf{I} & & & & \\ \frac{-t}{1}\mathbf{P} & \mathbf{I} & & & \\ & \frac{-t}{2}\mathbf{P} & \ddots & & \\ & & \ddots & \mathbf{I} & \\ & & & \frac{-t}{N}\mathbf{P} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \vdots \\ \vdots \\ \mathbf{v}_N \end{bmatrix} = \begin{bmatrix} \mathbf{s} \\ 0 \\ \vdots \\ \vdots \\ 0 \end{bmatrix}. \tag{7.9}$$

Let $\mathbf{v} = [\mathbf{v}_0; \mathbf{v}_1; \cdots; \mathbf{v}_N]$. An approximate solution $\hat{\mathbf{v}}$ to (7.9) would have block components $\hat{\mathbf{v}}_k$ such that $\sum_{k=0}^{N} \hat{\mathbf{v}}_k \approx T_N(t\mathbf{P})\mathbf{s}$, our desired approximation to $e^t\mathbf{h}$. In practice, we update only a single length n solution vector, adding all updates to that vector, instead of maintaining the $N+1$ different block vectors $\hat{\mathbf{v}}_k$ as part of $\hat{\mathbf{v}}$; furthermore, the block matrix and right-hand side are never formed explicitly.

With this block system in place, we can describe the algorithm's steps.

7.1.4 The hk-relax algorithm

Given a random walk transition matrix $\mathbf{P}$, scalar $t > 0$, and seed vector $\mathbf{s}$ as inputs, we solve the linear system from (7.9) as follows. Denote the initial solution vector by $\mathbf{y}$ and the initial $nN \times 1$ residual by $\mathbf{r}^{(0)} = \mathbf{e}_1 \otimes \mathbf{s}$. Denote by $r(i,j)$ the entry of $\mathbf{r}$ corresponding to node i in residual block j. The idea is to iteratively remove all entries from $\mathbf{r}$ that satisfy

$$r(i,j) \geq \frac{e^t \varepsilon d_i}{2N\psi_j(t)}. \tag{7.10}$$

To organize this process, we begin by placing the nonzero entries of $\mathbf{r}^{(0)}$ in a queue, $Q(\mathbf{r})$, and place updated entries of $\mathbf{r}$ into $Q(\mathbf{r})$ only if they satisfy (7.10).

Then `hk-relax` proceeds as follows.

1. At each step, pop the top entry of $Q(\mathbf{r})$, call it $r(i,j)$, and subtract that entry in $\mathbf{r}$, making $\mathbf{r}(i,j) = 0$.
2. Add $r(i,j)$ to $\mathbf{y}_i$.
3. Add $r(i,j)\frac{t}{j+1}\mathbf{P}\mathbf{e}_i$ to residual block $\mathbf{r}_{j+1}$.
4. For each entry of $\mathbf{r}_{j+1}$ that was updated, add that entry to the back of $Q(\mathbf{r})$ if it satisfies (7.10).

Once all entries of $\mathbf{r}$ that satisfy (7.10) have been removed, the resulting solution vector $\mathbf{y}$ will satisfy (7.3), which we prove in Section 7.2, along with a bound on the work required to achieve this. We have made our implementation of this publicly available. We show use of sparse data structures (e.g. hash tables to store the residual) make our algorithm highly efficient in practice.

7.1.5 Choosing N

The last detail of the algorithm we need to discuss is how to pick N. In (7.4) we want to guarantee the accuracy of $\mathbf{D}^{-1}\exp\{t\mathbf{P}\}\,\mathbf{s} - \mathbf{D}^{-1}T_N(t\mathbf{P})\mathbf{s}$. By using $\mathbf{D}^{-1}\exp\{t\mathbf{P}\} = \exp\left\{t\mathbf{P}^T\right\}\mathbf{D}^{-1}$ and $\mathbf{D}^{-1}T_N(t\mathbf{P}) = T_N(t\mathbf{P}^T)\mathbf{D}^{-1}$, we can get a new upperbound on $\|\mathbf{D}^{-1}\exp\{t\mathbf{P}\}\,\mathbf{s} - \mathbf{D}^{-1}T_N(t\mathbf{P})\mathbf{s}\|_\infty$ by noting

$$\begin{aligned}&\|\exp\left\{t\mathbf{P}^T\right\}\mathbf{D}^{-1}\mathbf{s} - T_N(t\mathbf{P}^T)\mathbf{D}^{-1}\mathbf{s}\|_\infty\\ &\quad\le \|\exp\left\{t\mathbf{P}^T\right\} - T_N(t\mathbf{P}^T)\|_\infty\|\mathbf{D}^{-1}\mathbf{s}\|_\infty.\end{aligned}$$

Since $\mathbf{s}$ is stochastic, we have $\|\mathbf{D}^{-1}\mathbf{s}\|_\infty \le \|\mathbf{D}^{-1}\mathbf{s}\|_1 \le 1$. From [Liou, 1966] we know that the norm $\|\exp\left\{t\mathbf{P}^T\right\} - T_N(t\mathbf{P}^T)\|_\infty$ is bounded by

$$\frac{\|t\mathbf{P}^T\|_\infty^{N+1}}{(N+1)!}\frac{(N+2)}{(N+2-t)} \le \frac{t^{N+1}}{(N+1)!}\frac{(N+2)}{(N+2-t)}. \tag{7.11}$$

So to guarantee (7.4), it is enough to choose N that implies $\frac{t^{N+1}}{(N+1)!}\frac{(N+2)}{(N+2-t)} < \varepsilon/2$. Such an N can be determined efficiently simply by iteratively computing terms of the Taylor polynomial for e^t until the error is less than the desired error for `hk-relax`. In practice, this required a choice of N no greater than $2t\log(\frac{1}{\varepsilon})$. Below we present a proof that a slightly larger choice of N will guarantee the desired accuracy.

Lemma 7.1.1 *Fix a constant $t \in [1,\infty)$ and an accuracy $\varepsilon \in (0,1)$. Then to guarantee that a Taylor polynomial $T_N(t)$ approximates e^t with error satisfying $|e^t - T_N(t)| < \varepsilon$ it suffices to choose the degree N to satisfy*

$$N \ge \max\{\ (\tfrac{2}{t}\log(\tfrac{1}{\varepsilon}) + \log(t) + 1)\lceil t\rceil,\ 3\lceil t\rceil\ \},$$

where log *is the natural logarithm.*

Proof As mentioned above, from [Liou, 1966] we know

$$|e^t - T_N(t)| < \frac{t^{N+1}}{(N+1)!}\frac{N+2}{N+2-t}.$$

If $N > 2t$ then the right-hand quantity in this inequality is bounded above by $\frac{t^N}{N!}$ (the proof follows from straight forward algebraic manipulation). For convenience we write $N = kT$, where we define $T = \lceil t\rceil$ assume that N is chosen so that k is an integer.

First we prove the following inequality, which will enable the rest of the proof:

$$\frac{t^{kT}}{(kT)!} < \frac{1}{T!}\left(\frac{t}{(k-1)!}\right)^T.$$

To prove the claim, we rewrite the expression:

$$\begin{aligned}\frac{t^{kT}}{(kT)!} &= \frac{t^T}{T!}\frac{t^T}{(2T)!/(T!)}\cdots\frac{t^T}{(kT)!/((k-1)T)!}\\ &< \frac{t^T}{T!}\left(\frac{1}{1}\right)^T\left(\frac{1}{2}\right)^T\cdots\left(\frac{1}{(k-1)}\right)^T\\ &= \frac{t^T}{T!}\left(\frac{1}{(k-1)!}\right)^T.\end{aligned}$$

Then rearranging the right-hand side to $(t/(k-1)!)^T/(T!)$ proves the claim.

Finally, to guarantee $|e^t - T_N(t)| < \varepsilon$, it suffices now to show that $\frac{1}{T!}\left(\frac{t}{(k-1)!}\right)^T < \varepsilon$. Rearranging and then taking log yields

$$\frac{t^T}{T!}\frac{1}{\varepsilon} < ((k-1)!)^T \tag{7.12}$$

$$T\log(t) + \log(\tfrac{1}{\varepsilon}) - \log(T!) < T\log((k-1)!). \tag{7.13}$$

To proceed we use the well-known inequality $m\log(m)/2 < \log(m!)$ on both sides of Inequality (7.13). We have that (7.13) is implied by

$$\begin{aligned}T\log(t) + \log(\tfrac{1}{\varepsilon}) - \tfrac{1}{2}T\log(T) &< T\log((k-1)!) && \text{simplify}\\ \log(t) + \tfrac{1}{T}\log(\tfrac{1}{\varepsilon}) - \tfrac{1}{2}\log(T) &< \log((k-1)!).\end{aligned}$$

Next note that, because $T = \lceil t\rceil \geq t$, the above inequality is implied by

$$\log(t) + \tfrac{1}{t}\log(\tfrac{1}{\varepsilon}) - \tfrac{1}{2}\log(t) < \log((k-1)!) \qquad \text{simplify} \tag{7.14}$$

$$\tfrac{1}{2}\log(t) + \tfrac{1}{t}\log(\tfrac{1}{\varepsilon}) < \log((k-1)!) \qquad \text{which is implied by} \tag{7.15}$$

$$\tfrac{1}{2}\log(t) + \tfrac{1}{t}\log(\tfrac{1}{\varepsilon}) < \tfrac{1}{2}(k-1)\log(k-1). \tag{7.16}$$

Finally, if $k-1 \geq e$ then $\log(k-1) \geq 1$, and so Inequality (7.16) is implied by assuming $k \geq \max\{\ 1 + \log(t) + 2\log(1/\varepsilon)/t,\ 1+e\ \}$. Recalling that $N = k\lceil t\rceil$, this completes the proof. ■

7.2 Convergence theory

We begin with an outline of the proof to give an intuition for the individual steps. First, we relate the error vector of the Taylor approximation $E_1 = T_N(t\mathbf{P})\mathbf{s} - \mathbf{x}$, to the error vector from solving the linear system described in Section 7.1.4, $E_2 = T_N(t\mathbf{P})\mathbf{s} - \mathbf{y}$. Second, we express the error vector E_2 in terms of the residual blocks of the linear system (7.9); this will involve writing E_2 as a sum of residual blocks $\mathbf{r}_k$ with weights $\psi_k(t\mathbf{P})$. Third, we use the previous results to upperbound $\|\mathbf{D}^{-1}T_N(t\mathbf{P})\mathbf{s} - \mathbf{D}^{-1}\mathbf{x}\|_\infty$ with $\sum_{k=0}^N \psi_k(t)\|\mathbf{D}^{-1}\mathbf{r}_k\|_\infty$, and use this to show that $\|\mathbf{D}^{-1}T_N(t\mathbf{P})\mathbf{s} - \mathbf{D}^{-1}\mathbf{x}\|_\infty < \varepsilon/2$ is guaranteed by the stopping criterion of `hk-relax`, (7.3). Finally,

we prove that performing steps of `hk-relax` until the stopping criterion is attained requires work bounded by $\frac{2N\psi_1(t)}{\varepsilon} \leq 2Ne^t/\varepsilon$.

Next we state our main result bounding the work required by `hk-relax` to approximate the heat kernel with accuracy as described in (7.3).

Theorem 7.2.1 *Let $\boldsymbol{P}$, t, $\psi_k(t)$, and $\boldsymbol{r}$ be as in Section 7.1. If steps of `hk-relax` are performed until all entries of the residual satisfy $r(i,j) < \frac{e^t \varepsilon d_i}{2N\psi_j(t)}$, then `hk-relax` produces an approximation $\boldsymbol{x}$ of $\boldsymbol{h} = \exp\{-t(\boldsymbol{I}-\boldsymbol{P})\}\,\boldsymbol{s}$ satisfying*

$$\|\boldsymbol{D}^{-1}\exp\{-t(\boldsymbol{I}-\boldsymbol{P})\}\,\boldsymbol{s} - \boldsymbol{D}^{-1}\boldsymbol{x}\|_\infty < \varepsilon,$$

and the amount of work required is bounded by

$$\textit{work}(\varepsilon) \leq \frac{2N\psi_1(t)}{\varepsilon} \leq \frac{2N(e^t-1)}{\varepsilon t}.$$

Recall from Lemma 7.1.1 that it suffices to choose $N = \max\{\ (\frac{2}{t}\log(\frac{1}{\varepsilon}) + \log(t) + 1)\lceil t\rceil,\ 3\lceil t\rceil\ \}$ to guarantee the desired accuracy.

Producing $\mathbf{x}$ satisfying

$$\|\mathbf{D}^{-1}\exp\{-t(\mathbf{I}-\mathbf{P})\}\,\mathbf{s} - \mathbf{D}^{-1}\mathbf{x}\|_\infty < \varepsilon$$

is equivalent to producing $\mathbf{y}$ satisfying

$$\|\mathbf{D}^{-1}\exp\{t\mathbf{P})\}\,\mathbf{s} - \mathbf{D}^{-1}\mathbf{y}\|_\infty < e^t\varepsilon.$$

We will show that the error vector in the `hk-relax` steps, $T_N(t\mathbf{P})\mathbf{s} - \mathbf{y}$, satisfies $\|\mathbf{D}^{-1}T_N(t\mathbf{P})\mathbf{s} - \mathbf{D}^{-1}\mathbf{y}\|_\infty < e^t\varepsilon/2$.

The following lemma expresses the error vector, $T_N(t\mathbf{P})\mathbf{s} - \mathbf{y}$, as a weighted sum of the residual blocks $\mathbf{r}_k$ in the linear system (7.9), and shows that the polynomials $\psi_k(t)$ are the weights.

Lemma 7.2.1 *Let $\psi_k(t)$ be defined as in Section 7.1. Then in the notation of Section 7.1.3, we can express the error vector of `hk-relax` in terms of the residual blocks $\boldsymbol{r}_k$ as follows*

$$T_N(t\boldsymbol{P})\boldsymbol{s} - \boldsymbol{y} = \sum_{k=0}^{N} \psi_k(t\boldsymbol{P})\boldsymbol{r}_k \tag{7.17}$$

Proof Consider (7.9). Recall that $\mathbf{v} = [\mathbf{v}_0; \mathbf{v}_1; \cdots; \mathbf{v}_N]$ and let $\mathbf{S}$ be the $(N+1)\times(N+1)$ matrix of 0s with first subdiagonal equal to $[\frac{1}{1}, \frac{1}{2}, \cdots, \frac{1}{N}]$. Then we can rewrite this linear system more conveniently as

$$(\mathbf{I} - \mathbf{S}\otimes(t\mathbf{P}))\mathbf{v} = \mathbf{e}_1 \otimes \mathbf{s}. \tag{7.18}$$

Let $\mathbf{v}_k$ be the true solution vectors that the $\hat{\mathbf{v}}_k$ are approximating in (7.18). We showed in Section 7.1.3 that the error $T_N(t\mathbf{P})\mathbf{s} - \mathbf{y}$ is in fact the sum of the errors $\mathbf{v}_k - \hat{\mathbf{v}}_k$. Now we will express $T_N(t\mathbf{P})\mathbf{s} - \mathbf{y}$ in terms of the residual partitions, i.e. $\mathbf{r}_k$.

At any given step we have $\mathbf{r} = \mathbf{e}_1 \otimes \mathbf{s} - (\mathbf{I} - \mathbf{S} \otimes (t\mathbf{P}))\hat{\mathbf{v}}$, so pre-multiplying by $(\mathbf{I} - \mathbf{S} \otimes (t\mathbf{P}))^{-1}$ yields $(\mathbf{I} - \mathbf{S} \otimes (t\mathbf{P}))^{-1}\mathbf{r} = \mathbf{v} - \hat{\mathbf{v}}$, because $(\mathbf{I} - \mathbf{S} \otimes (t\mathbf{P}))\mathbf{v} = \mathbf{e}_1 \otimes \mathbf{s}$ exactly, by definition of $\mathbf{v}$. Note that $(\mathbf{I} - \mathbf{S} \otimes (t\mathbf{P}))^{-1}\mathbf{r} = \mathbf{v} - \hat{\mathbf{v}}$ is the error vector for the linear system (7.18). From this, an explicit computation of the inverse (see Lemma 4.1.2 for details) yields

$$\begin{bmatrix} \mathbf{v}_0 - \hat{\mathbf{v}}_0 \\ \vdots \\ \mathbf{v}_N - \hat{\mathbf{v}}_N \end{bmatrix} = \left(\sum_{k=0}^{N} \mathbf{S}^k \otimes (t\mathbf{P})^k \right) \begin{bmatrix} \mathbf{r}_0 \\ \vdots \\ \mathbf{r}_N \end{bmatrix}. \tag{7.19}$$

For our purposes, the full block vectors $\mathbf{v}, \hat{\mathbf{v}}, \mathbf{r}$ and their individual partitions are unimportant: we want only their sum, because $T_N(t\mathbf{P})\mathbf{s} - \mathbf{y} = \sum_{k=0}^{N}(\mathbf{v}_k - \hat{\mathbf{v}}_k)$, as previously discussed.

Next we use (7.19) to express $\sum_{k=0}^{N}(\mathbf{v}_k - \hat{\mathbf{v}}_k)$, and hence

$$T_N(t\mathbf{P})\mathbf{s} - \mathbf{y},$$

in terms of the residual blocks $\mathbf{r}_k$. We accomplish this by examining the coefficients of an arbitrary block $\mathbf{r}_k$ in (7.19), which in turn requires analyzing the powers of $(\mathbf{S} \otimes t\mathbf{P})$.

Fix a residual block $\mathbf{r}_{j-1}$ and consider the product with a single term $(\mathbf{S}^k \otimes (t\mathbf{P})^k)$. Since $\mathbf{r}_{j-1}$ is in block-row j of $\mathbf{r}$, it multiplies with only the block-column j of each term $(\mathbf{S}^k \otimes (t\mathbf{P})^k)$, so we want to know what the blocks in block-column j of $(\mathbf{S}^k \otimes (t\mathbf{P})^k)$ look like.

From our previous analysis in Section 4.1.1 we know that

$$\mathbf{S}^m\mathbf{e}_j = \begin{cases} \frac{(j-1)!}{(j-1+m)!}\mathbf{e}_{j+m} & \text{if } 0 \le m \le N+1-j \\ 0 & \text{otherwise.} \end{cases}$$

This means that block-column j of $(\mathbf{S}^m \otimes (t\mathbf{P})^m)$ contains only a single nonzero block, $\frac{(j-1)!}{(j-1+m)!}t^m\mathbf{P}^m$, for all $0 \le m \le N+1-j$. Hence, summing the $n \times n$ blocks in block-column j of each power $(\mathbf{S}^m \otimes (t\mathbf{P})^m)$ yields

$$\sum_{m=0}^{N+1-j} \frac{(j-1)!t^m}{(j-1+m)!}\mathbf{P}^m \tag{7.20}$$

as the matrix coefficient of $\mathbf{r}_{j-1}$ in the right-hand side expression of $\sum_{k=0}^{N}(\mathbf{v}_k - \hat{\mathbf{v}}_k) = (\mathbf{e}^T \otimes \mathbf{I})(\sum_{m=0}^{N} \mathbf{S}^m \otimes (t\mathbf{P})^m)\mathbf{r}$.

Substituting $k = j-1$ on the right-hand side of (7.20) yields

$$\begin{aligned} \sum_{k=0}^{N}(\mathbf{v}_k - \hat{\mathbf{v}}_k) &= (\mathbf{e}^T \otimes \mathbf{I}) \left(\sum_{m=0}^{N} \mathbf{S}^m \otimes (t\mathbf{P})^m \right) \mathbf{r} \\ &= \sum_{k=0}^{N} \left(\sum_{m=0}^{N-k} \frac{k!t^m}{(k+m)!}\mathbf{P}^m \right) \mathbf{r}_k \\ &= \sum_{k=0}^{N} \psi_k(t\mathbf{P})\mathbf{r}_k, \end{aligned}$$

as desired. ■

Now that we've shown $T_N(t\mathbf{P})\mathbf{s} - \mathbf{y} = \sum_{k=0}^{N} \psi_k(t\mathbf{P})\mathbf{r}_k$ we can upperbound $\|\mathbf{D}^{-1}T_N(t\mathbf{P})\mathbf{s} - \mathbf{D}^{-1}\mathbf{y}\|_\infty$, to complete the proof of the first part of Theorem 7.2.1. To do this, we first show that

$$\|\mathbf{D}^{-1}\psi_k(t\mathbf{P})\mathbf{r}_k\|_\infty \leq \psi_k(t)\|\mathbf{D}^{-1}\mathbf{r}_k\|_\infty. \tag{7.21}$$

To prove this claim, recall that $\mathbf{P} = \mathbf{A}\mathbf{D}^{-1}$. Since $\psi_k(t)$ is a polynomial, we have $\mathbf{D}^{-1}\psi_k(t\mathbf{P}) = \mathbf{D}^{-1}\psi_k(t\mathbf{A}\mathbf{D}^{-1}) = \psi_k(t\mathbf{D}^{-1}\mathbf{A})\mathbf{D}^{-1}$, which equals $\psi_k(t\mathbf{P}^T)\mathbf{D}^{-1}$.

Taking the infinity norm and applying the triangle inequality to the definition of $\psi_k(t)$ gives us

$$\|\mathbf{D}^{-1}\psi_k(t\mathbf{P})\mathbf{r}_k\|_\infty \leq \sum_{m=0}^{N-k} \frac{t^m k!}{(m+k)!}\|(\mathbf{P}^T)^m(\mathbf{D}^{-1}\mathbf{r}_k)\|_\infty.$$

Then we have $\|(\mathbf{P}^T)^m(\mathbf{D}^{-1}\mathbf{r}_k)\|_\infty \leq \|(\mathbf{P}^T)^m\|_\infty\|(\mathbf{D}^{-1}\mathbf{r}_k)\|_\infty$ which equals $\|(\mathbf{D}^{-1}\mathbf{r}_k)\|_\infty$ because $\mathbf{P}^T$ is row-stochastic. Returning to our original objective, we then have

$$\begin{aligned}\|\mathbf{D}^{-1}\psi_k(t\mathbf{P})\mathbf{r}_k\|_\infty &\leq \sum_{m=0}^{N-k} \frac{t^m k!}{(m+k)!}\|(\mathbf{D}^{-1}\mathbf{r}_k)\|_\infty \\ &= \psi_k(t)\|\mathbf{D}^{-1}\mathbf{r}_k\|_\infty,\end{aligned}$$

proving the claim in (7.21).

This allows us to continue:

$$\begin{aligned} T_N(t\mathbf{P})\mathbf{s} - \mathbf{y} &= \sum_{k=0}^{N} \psi_k(t\mathbf{P})\mathbf{r}_k && \text{so} \\ \|\mathbf{D}^{-1}T_N(t\mathbf{P})\mathbf{s} - \mathbf{D}^{-1}\mathbf{y}\|_\infty &\leq \sum_{k=0}^{N} \|\mathbf{D}^{-1}\psi_k(t\mathbf{P})\mathbf{r}_k\|_\infty \\ &\leq \sum_{k=0}^{N} \psi_k(t)\|\mathbf{D}^{-1}\mathbf{r}_k\|_\infty. \end{aligned}$$

The stopping criterion for `hk-relax` requires that every entry of the residual satisfies $r(i,j) < \frac{e^t\varepsilon d_i}{2N\psi_j(t)}$, which is equivalent to satisfying $\frac{r(i,j)}{d_i} < \frac{e^t\varepsilon}{2N\psi_j(t)}$. This condition guarantees that $\|\mathbf{D}^{-1}\mathbf{r}_k\|_\infty < \frac{e^t\varepsilon}{2N\psi_k(t)}$. Thus we have

$$\begin{aligned}\|\mathbf{D}^{-1}T_N(t\mathbf{P})\mathbf{s} - \mathbf{D}^{-1}\mathbf{y}\|_\infty &\leq \sum_{k=0}^{N} \psi_k(t)\|\mathbf{D}^{-1}\mathbf{r}_k\|_\infty \\ &\leq \sum_{k=0}^{N} \left(\psi_k(t)\frac{e^t\varepsilon}{2N\psi_k(t)}\right)\end{aligned}$$

which is bounded above by $e^t\varepsilon/2$. Finally, we have that

$$\|\mathbf{D}^{-1}T_N(t\mathbf{P})\mathbf{s} - \mathbf{D}^{-1}\mathbf{y}\|_\infty < e^t\varepsilon/2$$

implies $\|\mathbf{D}^{-1}(\exp\{-t(\mathbf{I}-\mathbf{P})\}\,\mathbf{s} - \mathbf{x})\|_\infty < \varepsilon/2$, completing our proof of the first part of Theorem 7.2.1.

Bounding work It remains to bound the work required to perform steps of `hk-relax` until the stopping criterion is achieved.

Because the stopping criterion requires each residual entry to satisfy

$$\frac{r(i,j)}{d_i} < \frac{e^t \varepsilon}{2N\psi_j(t)}$$

we know that each step of `hk-relax` must operate on an entry $r(i,j)$ larger than this threshold. That is, each step we relax an entry of $\mathbf{r}$ satisfying $\frac{r(i,j)}{d_i} \geq \frac{e^t \varepsilon}{2N\psi_j(t)}$.

Consider the solution vector $\mathbf{y} \approx T_N(t\mathbf{P})\mathbf{s}$ and note that each entry of $\mathbf{y}$ is really a sum of values that we have deleted from $\mathbf{r}$. But $\mathbf{r}$ always contains only nonnegative values: the seed vector $\mathbf{s}$ is nonnegative, and each step involves setting entry $r(i,j) = 0$ and adding a scaled column of $\mathbf{P}$, which is nonnegative, to $\mathbf{r}$. Hence, $\|\mathbf{y}\|_1$ equals the sum of the $r(i,j)$ added to $\mathbf{y}$, $\sum_{l=1}^{T} r(i_l, j_l) = \|\mathbf{y}\|_1$.

Finally, since the entries of $\mathbf{y}$ are nondecreasing (because they only change if we add positive values $r(i,j)$ to them), we know that $\|\mathbf{y}\|_1 \leq \|T_N(t\mathbf{P})\mathbf{s}\|_1 = \psi_0(t) \leq e^t$. This implies, then, that

$$\sum_{l=1}^{T} r(i_l, j_l) \leq e^t.$$

Using the fact that the values of $r(i,j)$ that we relax must satisfy $\frac{r(i,j)}{d_i} \geq \frac{e^t \varepsilon}{2N\psi_j(t)}$ we have that

$$\sum_{l=1}^{T} \frac{d_{i_l} e^t \varepsilon}{2N\psi_{j_l}(t)} \leq e^t.$$

Simplifying yields

$$\sum_{l=1}^{T} \frac{d_{i_l}}{\psi_{j_l}(t)} \leq \frac{2N}{\varepsilon}.$$

By Lemma 7.2.1 we know $\psi_k(t) \leq \psi_1(t)$ for each $k \geq 1$, so we can lowerbound $\sum_{l=1}^{T} \frac{d_{i_l}}{\psi_1(t)} \leq \sum_{l=1}^{T} \frac{d_{i_l}}{\psi_{j_l}(t)} \leq \frac{2N}{\varepsilon}$, giving us

$$\sum_{l=1}^{T} d_{i_l} \leq \frac{2N\psi_1(t)}{\varepsilon}.$$

Finally, note that the dominating suboperation in each step of `hk-relax` consists of relaxing $r(i,j)$ and spreading this "heat kernel rank" to the neighbors of node i in block $\mathbf{r}_{j+1}$. Since node i has d_i neighbors, this step consists of d_i adds, and so the work performed by `hk-relax` is $\sum_{l=1}^{T} d_{i_l}$, which is exactly the quantity we bounded above.

7.2.1 Fast algorithm for diagonal entries of the heat kernel

Using the diagonal entries of the heat kernel, $\exp\{-t(\mathbf{I} - \mathbf{P})\}_{jj}$, has been proposed to measure the ability of information to diffusion through certain nodes in a graph, via the subgraph centrality

metric [Benzi and Klymko, 2013]. Previously the best known method for obtaining all n such diagonal entries [Gene H. Golub, 2010] required on the order of $O(n^2)$ work as it relied on dense matrix vector products inside of neat applications of Guass quadrature and the Lanczos method on the underlying matrix.

We remark here that our method enables computation of all n diagonal entries $h_j = \exp\{-(\mathbf{I}-\mathbf{P})\}_{jj}$ with a desired point-wise accuracy of ε, in work bounded by $O(|E|)$. In other words, for all j we can use our `hk-relax` method to compute $\hat{h}_j$ satisfying $|h_j - \hat{h}_j| \leq \varepsilon$, and the total amount of work involved is bounded by $O(|E|/\varepsilon)$. For each j, simply use our `hk-relax` method to compute the full diffusion vector $\mathbf{h}$ to a degree-normalized infinite-norm accuracy of $\varepsilon/d(j)$. By the work bound obtained in the previous section, this can be done for node j in work bounded by $O(Ne^t/(\varepsilon/d(j)))$, and ignoring the constants e^t and N, this is equivalent to $O(d(j)/\varepsilon)$. Summing this quantity for all nodes j yields a total bound on work of $O(\sum_{j=1}^{n} d(j)/\varepsilon) = O(|E|/\varepsilon)$.

This difference in runtime, i.e. from $O(n^2)$, is an improvement for any graph that is sparse enough that $|E|$ is significantly smaller than n^2, and this improvement comes without any optimization of our `hk-relax` method specifically toward this task of computing these diagonal entries. One work in progress following this thesis is to more intelligently adapt our `hk-relax` method for the specific computation of the diagonal entries of the heat kernel and other graph diffusions and matrix functions. Next we move on to an experimental evaluation of the heat kernel algorithm we have just analyzed.

7.3 Experimental Results

Here we compare our `hk-relax` with a PageRank-based local clustering algorithm, pprpush [Andersen et al., 2006a]. Both algorithms accept as inputs a symmetric graph $\mathbf{A}$ and seed set $\mathbf{s}$. The parameters required are t and ε, for `hk-relax`, and α and ε for pprpush. Both algorithms compute their respective diffusion ranks starting from the seed set, then perform a sweep-cut on the resulting ranks. The difference in the algorithms lies solely in the diffusion used and the particular parameters. We conducted the timing experiments on a Dual CPU system with the Intel Xeon E5-2670 processor (2.6 GHz, 8 cores) with 16 cores total and 256 GB of RAM. None of the experiments needed anywhere near all the memory, nor any of the parallelism. Our implementation uses Matlab's sparse matrix data structure through a C++ mex interface. It uses C++ unordered maps to store sparse vectors.

As a general comment, the experimental results below are what would be expected from Figure 7.1: PageRank diffuses to a larger region of the graph whereas the heat kernel remains more focused in a sub-region. PageRank, then, finds a large community with thousands of nodes whereas the heat kernel finds a small community with only a few hundred nodes with slightly worse conductance. This experiment suggests that, if these results also hold in real-world networks, then because real-world

communities are often small [Leskovec et al., 2009], the heat kernel diffusion should produce *more accurate* communities in real-world networks.

7.3.1 Runtime and conductance

Here we compare the runtime and conductance of the algorithms on a suite of social networks. For pprpush, we fix $\alpha = 0.99$, then compute PageRank for multiple values of $\varepsilon = 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$, and output the set of best conductance obtained. (This matches the way this method is commonly used in past work.) For `hk-relax`, we compute the heat kernel rank for four different parameter sets, and output the set of best conductance among them: $(t, \varepsilon) = (10, 10^{-4}); (20, 10^{-3}); (40, 5 \cdot 10^{-3}); (80, 10^{-2})$. We also include in `hk-relax` an early termination criterion, in the case that the sum of the degrees of the nodes which are relaxed, $\sum d_{i_l}$, exceeds $n^{1.5}$. However, even the smaller input graphs (on which the condition is more likely to be met because of the smaller value of $n^{1.5}$) do not appear to have reached this threshold. Furthermore, the main theorem of this paper implies that the quantity $\sum d_{i_l}$ cannot exceed $\frac{2N\psi_1(t)}{\varepsilon}$. The datasets we use are summarized in Table 7.2; all datasets are modified to be undirected and a single connected component. These datasets were originally presented in the following papers [Alberich et al., 2002, Boguñá et al., 2004, Leskovec et al., 2007, Newman, 2006, 2001, Leskovec et al., 2009, 2010, (The Cooperative Association for Internet Data Analyais), 2005, Kwak et al., 2010, Yang and Leskovec, 2012, Boldi et al., 2011, Mislove et al., 2007].

To compare the runtimes of the two algorithms, we display in Figure 7.1 (top) for each graph the 25%, 50%, and 75% percentiles of the runtimes from 200 trials performed. For a given graph, each trial consisted of choosing a node of that graph uniformly at random to be a seed, then calling both the PageRank and the heat kernel algorithms. On the larger datasets, which have a much broader spectrum of node degrees and therefore greater variance, we instead performed 1,000 trials. Additionally, we display in Figure 7.1 (bottom) the 25%, 50%, and 75% percentiles of the conductances achieved during the exact same set of trials. The trendlines of these figures omit some of the trials in order to better show the trends, but all of the median results are plotted (as open circles). The figures show that in small graphs, `hk-relax` is faster than pprpush, but gets larger (worse) conductance. The picture reverses for large graphs where `hk-relax` is slower but finds smaller (better) conductance sets.

Cluster size and conductance. We highlight the individual results of the previous experiment on the symmetrized twitter network. Here, we find that `hk-relax` finds sets of better conductance than pprpush at all sizes of communities in the network. In Figure 7.2, the top figure shows a scatter plot of conductance vs. community size in the twitter graph for the two community detection

Table 7.2.
Datasets for comparison of heat kernel and PageRank diffusions

Graph	$\|V\|$	$\|E\|$
pgp-cc	10,680	24,316
ca-AstroPh-cc	17,903	196,972
marvel-comics-cc	19,365	96,616
as-22july06	22,963	48,436
rand-ff-25000-0.4	25,000	56,071
cond-mat-2003-cc	27,519	116,181
email-Enron-cc	33,696	180,811
cond-mat-2005-fix-cc	36,458	171,735
soc-sign-epinions-cc	119,130	704,267
itdk0304-cc	190,914	607,610
dblp-cc	226,413	716,460
flickr-bidir-cc	513,969	3,190,452
ljournal-2008	5,363,260	50,030,085
twitter-2010	41,652,230	1,202,513,046
friendster	65,608,366	1,806,067,135
com-amazon	334,863	925,872
com-dblp	317,080	1,049,866
com-youtube	1,134,890	2,987,624
com-lj	3,997,962	34,681,189
com-orkut	3,072,441	117,185,083

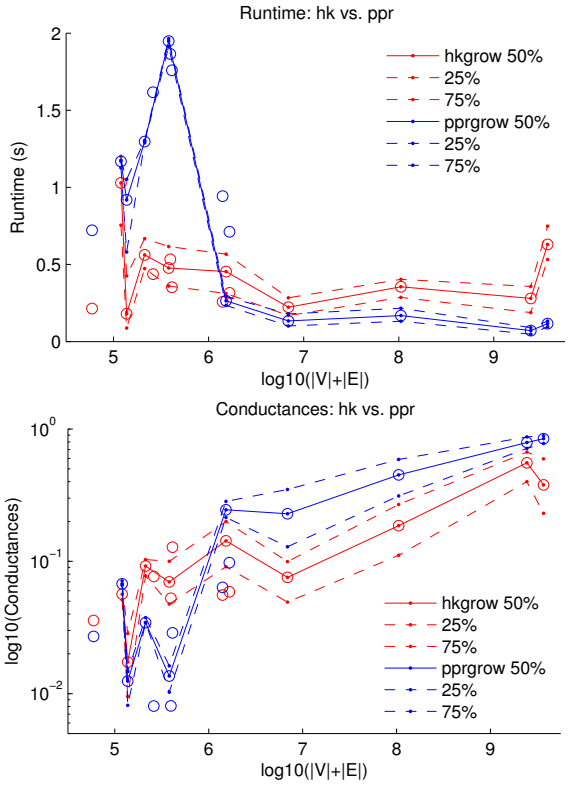

Figure 7.1. Comparing runtime and conductance of heat kernel and PageRank

methods; the bottom figure shows a kernel density estimate of the conductances achieved by each method, which shows that `hk-relax` is more likely to return a set of lower conductance.

7.3.2 Clusters produced vs. ground-truth

We conclude with an evaluation of the two diffusions' performance in a ground-truth community recovery experiment. We run both algorithms to identify ground truth communities in the `com-dblp`, `com-lj`, `com-amazon`, `com-orkut`, `com-youtube`, and `com-friendster` datasets [Yang and Leskovec, 2012, Mislove et al., 2007]. In this experiment, for each dataset we first located 100 known communities in the dataset of size greater than 10. Given one such community, using every single node as an individual seed, we looked at the sets returned by `hk-relax` with $t = 5, \varepsilon = 10^{-4}$ and pprpush using the standard procedure. We picked the set from the seed that had the highest F_1 measure. (Recall that the F_1 measure is a harmonic mean of precision and recall.) We report the mean of the F_1 measure, conductance, and set size, where the average is taken over all 100 trials in Table 7.3. These results show that `hk-relax` produces only slightly inferior conductance scores, but using much

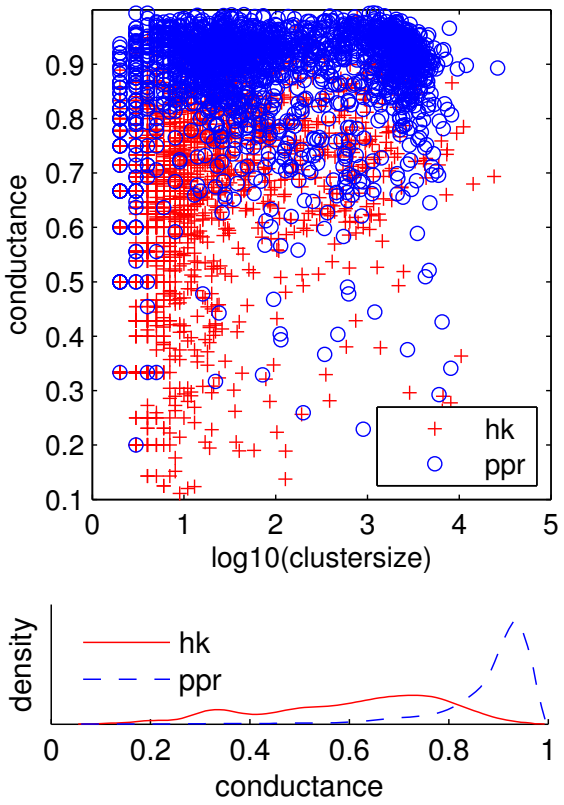

Figure 7.2. Comparing cluster size and conductance produced by the heat kernel and PageRank

Table 7.3.
Heat kernel vs PageRank in ground-truth community detection

data	F_1-measure		conductance		set size	
	`hk`	`pr`	`hk`	`pr`	`hk`	`pr`
`amazon`	0.608	0.415	0.124	0.050	145	5,073
`dblp`	0.364	0.273	0.238	0.144	156	4,529
`youtube`	0.128	0.078	0.477	0.361	137	5,833
`lj`	0.138	0.104	0.564	0.524	156	299
`orkut`	0.183	0.116	0.824	0.736	95	476
`friendster`	0.125	0.112	0.866	0.860	96	218

smaller sets with substantially better F_1 measures. This suggests that `hk-relax` better captures the properties of real-world communities than the PageRank diffusion in the sense that the tighter sets

produced by the heat kernel are better focused around real-world communities than are the larger sets produced by the PageRank diffusion.

Finally, we thank AmirMahdi Ahmadinejad for pointing out a small mistake in our original implementation of this experiment – though we reported in Table 7.3 in our original publication [Kloster and Gleich, 2014] that we carried out this experiment on 100 commmunities of size at least 10, AmiMahdi noted that in practiced we had used some communities of size less than 10. The results shown here in Table 7.3 are from the corrected experiment, and in fact are even more favorable for our `hk-relax` algorithm than our original reported results.

7.4 General diffusions in constant time

In this section we adapt the framework presented in Section 7.1 for computing a local heat kernel diffusion to compute any graph diffusion. We then show that the general framework can be understood as a generalization of PageRank that uses a matrix-valued teleportation parameter instead of the standard scalar-valued parameter $\alpha \in (0, 1)$ (Section 7.4.1). Finally, we prove that our general diffusion algorithm computes an approximation with degree-normalized infinity error bounded by ε, using an amount of work bounded by $O(N^2/\varepsilon)$, where N is the number of diffusion coefficients required to approximate the diffusion with an accuracy of ε (Theorem 7.4.1).

We remark that, although we present here a deterministic, constant-time algorithm for computing any diffusion, it is not necessarily the case that a sweep-cut over any such diffusion will yield a cluster with good conductance. It is known that PageRank [Andersen et al., 2006b] and the heat kernel [Chung, 2007b] have Cheeger inequalities – that is, a theoretical guarantee that the conductance obtained by a sweep-cut over the given diffusion is bounded by some function of the optimal conductance. Currently it is an open question whether there exists a Cheeger inequality relating the conductance obtained by a sweep-cut over a general diffusion to the optimal conductance set near the seed set.

7.4.1 Constructing the general diffusion linear system

Let $\mathbf{f}$ be a general diffusion. That is, let c_k be coefficients satisfying $c_k \geq 0$, $\sum_{k=0}^{\infty} c_k = 1$, and define

$$\mathbf{f} = \sum_{k=0}^{\infty} c_k \mathbf{P}^k \mathbf{s}$$

for some seed vector $\mathbf{s}$. We will construct a linear system very closely related to the one presented in Section 7.1 and then derive our algorithm from it.

The diffusion we want to compute, $\mathbf{f} = \sum_{k=0}^{\infty} c_k \mathbf{P}^k \mathbf{s}$, has terms we denote by $\mathbf{v}_k := c_k \mathbf{P}^k \mathbf{s}$. Define the countably-dimensioned vector $\mathbf{c} = \begin{bmatrix} c_0 & c_1 & \cdots \end{bmatrix}^T$. Next we construct a linear system from which we can compute the desired diffusion $\mathbf{f}$. Set the matrix $\mathbf{S}$ to be the matrix with countably-infinite rows and columns, where the entries of $\mathbf{S}$ are all 0s, except with 1s on the first subdiagonal, i.e. $\mathbf{S}_{j+1,j} = 1$ for $j = 1, 2, \cdots$.

$$\begin{bmatrix} \mathbf{I} & & & & & \\ -\mathbf{P} & \mathbf{I} & & & & \\ & -\mathbf{P} & \mathbf{I} & & & \\ & & -\mathbf{P} & \mathbf{I} & & \\ & & & \ddots & \ddots & \end{bmatrix} \begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \vdots \end{bmatrix} = \begin{bmatrix} \mathbf{s} \\ 0 \\ 0 \\ \vdots \\ \vdots \end{bmatrix}. \tag{7.22}$$

Each diagonal block is an $n \times n$ identity matrix $\mathbf{I}$, and each vector $\mathbf{v}_j$ is an $n \times 1$ vector. Denote the vector $[\mathbf{v}_0^T, \mathbf{v}_1^T, \mathbf{v}_2^T, \cdots]^T$ by $\mathbf{x}$, and note that the right-hand side of Equation (7.22) can be expressed as $\mathbf{e}_1 \otimes \mathbf{s}$. Finally, note that the matrix in Equation (7.22) can be expressed as $\mathbf{I} - \mathbf{S} \otimes \mathbf{P}$, which allows us to compactly express the system in (7.22) as

$$(\mathbf{I} - \mathbf{S} \otimes \mathbf{P})\mathbf{x} = (\mathbf{e}_1 \otimes \mathbf{s}). \tag{7.23}$$

Assuming the solution, $\mathbf{x}$, to (7.22) is attained,we can then recover the diffusion $\mathbf{f}$ via

$$\mathbf{f} = \sum_{k=0}^{\infty} c_k \mathbf{v}_k = (\mathbf{c} \otimes \mathbf{I})^T \begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \vdots \end{bmatrix} \tag{7.24}$$

where $\mathbf{I}$ is the $n \times n$ identity matrix. In Section 7.4.5 below we show how to adapt our algorithm from Section 7.1 to compute a general diffusion using Equation (7.24).

A generalization of PageRank

While the form in (7.24) is useful for convergence analysis (Section 7.4.2), it leaves us unsatisfied in that it is just shy of generalizing the PageRank equation to have a matrix-valued teleportation parameter in place of the scalar-valued teleportation parameter α. We demonstrate here via a simple similarity transformation that the diffusion in (7.24) is in fact equivalent to a vector constructed from the solution to a matrix-valued PageRank system, under the condition that all coefficients c_j are nonzero.

Let $\mathbf{C}$ denote the diagonal matrix with c_{j-1} on the jth entry, $\mathbf{C}_{jj}$. Under the assumption that all c_j are nonozero, $\mathbf{C}$ is invertible, so we can define $\Gamma = \mathbf{CSC}^{-1}$. Note that this is simply the matrix of all zeros with first subdiagonal equal to $[c_1/c_0, c_2/c_1, \cdots]$. Finally, we demonstrate that pre-multiplying Equation (7.24) by $(\mathbf{C} \otimes \mathbf{I})$ will produce a PageRank-type equation:

$$(\mathbf{C} \otimes \mathbf{I})(\mathbf{I} - \mathbf{S} \otimes \mathbf{P})\mathbf{x} = (\mathbf{C} \otimes \mathbf{I})(\mathbf{e}_1 \otimes \mathbf{s}) \tag{7.25}$$

$$(\mathbf{C} \otimes \mathbf{I})(\mathbf{I} - \mathbf{S} \otimes \mathbf{P})(\mathbf{C} \otimes \mathbf{I})^{-1}(\mathbf{C} \otimes \mathbf{I})\mathbf{x} = (c_0\mathbf{e}_1 \otimes \mathbf{s}) \tag{7.26}$$

$$(\mathbf{I} - (\mathbf{CSC}^{-1}) \otimes \mathbf{P})(\mathbf{C} \otimes \mathbf{I})\mathbf{x} = (c_0\mathbf{e}_1 \otimes \mathbf{s}) \tag{7.27}$$

$$(\mathbf{I} - \Gamma \otimes \mathbf{P})\mathbf{y} = (c_0\mathbf{e}_1 \otimes \mathbf{s}), \tag{7.28}$$

where $\mathbf{y} = (\mathbf{C} \otimes \mathbf{I})\mathbf{x}$. Equation (7.28) is simply the standard PageRank linear system, $(\mathbf{I} - \alpha\mathbf{P})\mathbf{x} = \mathbf{s}$, but with a matrix-valued teleportation parameter Γ in place of the standard scalar $\alpha \in (0, 1)$. We remark that our construction of Γ enables a solution to Equation (7.28) that is analogous to the conventional solution to the standard PageRank problem, namely via a Neumann series. Because α is chosen in $(0, 1)$ the standard PageRank problem can be solved via $(\mathbf{I} - \alpha\mathbf{P})^{-1} = \sum_{k=0}^{\infty} \alpha^k\mathbf{P}^k$. The lower-bidiagonal structure of Γ guarantees the existence of an analogous Neumann series inverse in the matrix-valued case: $(\mathbf{I} - \Gamma \otimes \mathbf{P})^{-1} = \sum_{k=0}^{\infty}(\Gamma \otimes \mathbf{P})^k = \sum_{k=0}^{\infty}(\Gamma^k \otimes \mathbf{P}^k)$.

This construction is not useful in computation (as it is known a prior that $\mathbf{y} = [\mathbf{v}_0^T, \mathbf{v}_1^T, \mathbf{v}_2^T, \cdots]^T$ is the solution to the system, and the vectors $\mathbf{v}_j = c_k\mathbf{P}^k\mathbf{s}$ can easily be computed). Rather, it is interesting as a point for future study: we have shown that the standard PageRank vector can be viewed as a special case of this matrix-valued system, and suspect that alternative solutions to the matrix-valued system (i.e. solutions derived using a variety of matrices Γ with different structures) could yield valuable information distinct from the standard PageRank vector. Although we do not further explore this formulation in the rest of this thesis, it is interesting enough in its own right to merit inclusion.

7.4.2 Solving the large linear system

Next we look at how to apply the Gauss-Southwell method in an efficient way to the above system, again following an approach related to that in [Kloster and Gleich, 2014]. To apply the Gauss-Southwell update to Equation (7.23) we make a modification to exploit the Kronecker structure in Equation (7.22). In particular, the standard residual update

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - r \cdot \mathbf{e}_j + r\mathbf{M}\mathbf{e}_j$$

we rewrite as follows. First, instead of having a linear index i.e. $\mathbf{e}_j$, the max magnitude of the entry in the (infinite-dimensioned) residual can instead be identified via a pair of indices: one index, j,

indicates which block of the residual the entry is in; the other index, i, indicates which entry of that residual block the desired element is in. For example, the solution vector $\mathbf{x}$ in Equation (7.22) has a block labelled $\mathbf{v}_2$. To operate on entry j of block $\mathbf{v}_2$, we can use the standard basis vector $\mathbf{e}_3 \otimes \mathbf{e}_j$ as follows:

$$(\mathbf{v}_2)_j = (\mathbf{e}_j)^T \mathbf{v}_2 = (\mathbf{e}_3 \otimes \mathbf{e}_j)^T \mathbf{x}.$$

Similarly, we use notation like $(\mathbf{e}_i \otimes \mathbf{e}_j)^T \mathbf{r}$ to refer to entry j of residual block i, when doing Gauss-Southwell updates on the iterative solution and residual vectors. So the standard Gauss-Southwell updates (Section 2.2), when applied to (7.22), become

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + r \cdot \mathbf{e}_i \otimes \mathbf{e}_j \tag{7.29}$$

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - r \cdot \mathbf{M}(\mathbf{e}_i \otimes \mathbf{e}_j) \tag{7.30}$$

$$= \mathbf{r}^{(k)} - r \cdot (\mathbf{I} - \mathbf{S} \otimes \mathbf{P})(\mathbf{e}_i \otimes \mathbf{e}_j) \tag{7.31}$$

$$= \mathbf{r}^{(k)} - r\mathbf{e}_i \otimes \mathbf{e}_j + r(\mathbf{S}\mathbf{e}_i \otimes \mathbf{P}\mathbf{e}_j) \tag{7.32}$$

$$= \mathbf{r}^{(k)} - r\mathbf{e}_i \otimes \mathbf{e}_j + (r\mathbf{e}_{i+1} \otimes \mathbf{P}\mathbf{e}_j) \qquad \text{since } \mathbf{S}\mathbf{e}_i = \mathbf{e}_{i+1} \tag{7.33}$$

This simply corresponds to adding $r\mathbf{e}_j$ to block i of the solution vector, subtracting $r\mathbf{e}_j$ from block i of the residual vector, and finally adding $\mathbf{P}\mathbf{e}_j$ to block $i+1$ of the residual vector. This is essentially the exact same as one iteration of the `hk-relax` algorithm presented in Section 7.1: updating one entry of the solution vector, and adding a scaled column of $\mathbf{P}$ to the residual. To solve the equation (i.e. compute the diffusion), just repeat this process until the degree-weighted infinity norm of the residual is below the tolerance ε. This is the same as in the original push framework, except bounding the residual's degree-weighted infinity norm is more complicated.

7.4.3 Error in terms of residuals

The proofs of accuracy and runtime in Section 7.1 require bounding quantities related to the residual vector. We adapt those proofs here to bound the error and the work for this general diffusion framework.

First, consider Equation (7.24). The actual diffusion vector that we want, $\mathbf{f}$, is related to the infinite linear system defined in (7.22) via the transformation $(\mathbf{c} \otimes \mathbf{I})^T$ discussed above. Now that we are producing an approximation $\hat{\mathbf{f}}$ to the diffusion $\mathbf{f}$, we want some to have some control over the error vector, $\mathbf{f} - \hat{\mathbf{f}}$.

It turns out we can use this same transformation, $(\mathbf{c} \otimes \mathbf{I})^T$ in Equation 7.24, to express the error vector $\mathbf{f} - \hat{\mathbf{f}}$ in terms of the blocks of the infinite residual vector, $\mathbf{r} = [\mathbf{r}_0, \mathbf{r}_1, \cdots]$, as follows. The

actual solution is $\mathbf{f} = \sum c_k \mathbf{v}_k$; denote the terms of our approximation by $\hat{\mathbf{v}}_k \approx \mathbf{v}_k$. Then $\hat{\mathbf{f}} = \sum c_k \hat{\mathbf{v}}_k$, and the error of our approximation can be expressed

$$\mathbf{f} - \hat{\mathbf{f}} = \sum_{k=0}^{\infty} c_k(\mathbf{v}_k - \hat{\mathbf{v}}_k) = (\mathbf{c} \otimes \mathbf{I})^T \left(\begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \vdots \end{bmatrix} - \begin{bmatrix} \hat{\mathbf{v}}_0 \\ \hat{\mathbf{v}}_1 \\ \vdots \end{bmatrix} \right). \tag{7.34}$$

To proceed, first we relate the error vector to the residual vector via a standard trick:

$$\begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \vdots \end{bmatrix} - \begin{bmatrix} \hat{\mathbf{v}}_0 \\ \hat{\mathbf{v}}_1 \\ \vdots \end{bmatrix} = (\mathbf{I} - \mathbf{S} \otimes \mathbf{P})^{-1} \begin{bmatrix} \mathbf{r}_0 \\ \mathbf{r}_1 \\ \vdots \end{bmatrix}. \tag{7.35}$$

Now left-multiply by $(\mathbf{c} \otimes \mathbf{I})^T$...

$$\mathbf{f} - \hat{\mathbf{f}} = (\mathbf{c} \otimes \mathbf{I})^T \left(\begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \vdots \end{bmatrix} - \begin{bmatrix} \hat{\mathbf{v}}_0 \\ \hat{\mathbf{v}}_1 \\ \vdots \end{bmatrix} \right) = (\mathbf{c} \otimes \mathbf{I})^T (\mathbf{I} - \mathbf{S} \otimes \mathbf{P})^{-1} \begin{bmatrix} \mathbf{r}_0 \\ \mathbf{r}_1 \\ \vdots \end{bmatrix}. \tag{7.36}$$

To be able to bound the error on an approximation, we will use the fact that $(\mathbf{I} - \mathbf{S} \otimes \mathbf{P})$ has inverse $\sum_{k=0}^{\infty} (\mathbf{S} \otimes \mathbf{P})^k$. To justify this fact, note that even though $\mathbf{S}$ is countably-dimensioned, the product $\mathbf{M} = (\mathbf{I} - \mathbf{S} \otimes \mathbf{P})(\sum_{k=0}^{\infty} (\mathbf{S} \otimes \mathbf{P})^k)$ telescopes, so that inspection of any single element $\mathbf{M}_{ij}$ with $i \neq j$ reveals it is zero. Thus, we can substitute in the explicit inverse:

$$\mathbf{f} - \hat{\mathbf{f}} = (\mathbf{c} \otimes \mathbf{I})^T \left(\sum_{k=0}^{\infty} (\mathbf{S} \otimes \mathbf{P})^k \right) \begin{bmatrix} \mathbf{r}_0 \\ \mathbf{r}_1 \\ \vdots \end{bmatrix}. \tag{7.37}$$

We want to expand this sum in terms of the vectors $\mathbf{r}_j$ so that we have an expression of the error in terms of the residual blocks. This requires collecting together all sub-blocks of the matrices $(\mathbf{S} \otimes \mathbf{P})^k$ that align with a particular residual block. For example, note that $\mathbf{r}_j$ will only multiply with sub-blocks of $(\mathbf{S} \otimes \mathbf{P})^k$ corresponding to column $(j+1)$ of $\mathbf{S}$. More specifically, note that $(\mathbf{S} \otimes \mathbf{P})^k = (\mathbf{S}^k) \otimes (\mathbf{P}^k)$, and that the only terms of the sum in (7.37) in which $\mathbf{r}_j$ will appear will be of the form $(\mathbf{S}^k)_{i,j+1} \cdot \mathbf{P}^k \mathbf{r}_j$. In fact, because of the sub-diagonal structure of $\mathbf{S}$, we can express $(\mathbf{S}^k)_{i,j+1} = 1$ iff $i = j + 1 + k$.

To proceed from here, we define a family of power series related to the original diffusion of interested. Let $\psi(x) = \sum_{k=0}^{\infty} c_k x^k$ be the diffusion we are interested in (note that these are closely related to the ψ functions defined in Section 7.1.2). Next, define

$$\psi_j(x) = \sum_{k=0}^{\infty} c_{j+k} x^k.$$

We are mostly interested in the quantities $\psi_j(1)$, which we abbreviate ψ_j. We will be able to apply the corollary and substitute in these functions ψ_j after some manipulation:

$$\mathbf{f} - \hat{\mathbf{f}} = (\mathbf{c} \otimes \mathbf{I})^T \left(\sum_{k=0}^{\infty} (\mathbf{S} \otimes \mathbf{P})^k \right) \begin{bmatrix} \mathbf{r}_0 \\ \mathbf{r}_1 \\ \vdots \end{bmatrix} \tag{7.38}$$

$$= \left(\sum_{k=0}^{\infty} (\mathbf{c}^T \mathbf{S}^k) \otimes (\mathbf{P}^k) \right) \begin{bmatrix} \mathbf{r}_0 \\ \mathbf{r}_1 \\ \vdots \end{bmatrix}, \tag{7.39}$$

and then multiplying the residual vector through yields

$$= \sum_{k=0}^{\infty} \left(\sum_{j=1}^{\infty} (\mathbf{c}^T \mathbf{S}^k)_j \mathbf{P}^k \mathbf{r}_{j-1} \right) \tag{7.40}$$

$$= \sum_{j=0}^{\infty} \left(\sum_{k=0}^{\infty} (\mathbf{c}^T \mathbf{S}^k)_{j+1} \mathbf{P}^k \right) \mathbf{r}_j. \qquad \text{after re-indexing.} \tag{7.41}$$

We note here that $(\mathbf{c}^T \mathbf{S}^k)_{j+1} = (\mathbf{S}^k)_{j+1+k,j+1} = c_{j+k}$. Substituting shows that

$$\mathbf{f} - \hat{\mathbf{f}} = \sum_{j=0}^{\infty} \left(\sum_{k=0}^{\infty} c_{j+k} \mathbf{P}^k \right) \mathbf{r}_j \tag{7.42}$$

$$= \sum_{j=0}^{\infty} \psi_j(\mathbf{P}) \mathbf{r}_j. \tag{7.43}$$

With this expression established, we can now study $\|\mathbf{D}^{-1}(\mathbf{f} - \hat{\mathbf{f}})\|_\infty$. in terms of the residuals at each stage and the weights $\psi_j(\mathbf{P})$. First, we derive

$$\mathbf{D}^{-1}(\mathbf{f} - \hat{\mathbf{f}}) \quad = \quad \sum_{j=0}^{\infty} \mathbf{D}^{-1} \psi_j(\mathbf{P}) \mathbf{r}_j \quad = \quad \sum_{j=0}^{\infty} \psi_j(\mathbf{P}^T) \mathbf{D}^{-1} \mathbf{r}_j.$$

To continue, recall that $\psi_j(x)$ is a power series with all positive coefficients, and $\mathbf{P}$ has all nonnegative entries. This enables us to bound $\|\psi_j(\mathbf{P}^T)\|_\infty \leq \psi_j(\|\mathbf{P}^T\|_\infty)$. So we can bound the error as follows:

$$\|\mathbf{D}^{-1}(\mathbf{f} - \hat{\mathbf{f}})\|_\infty = \left\| \sum_{j=0}^{\infty} \psi_j(\mathbf{P}^T) \mathbf{D}^{-1} \mathbf{r}_j \right\|_\infty \tag{7.44}$$

$$\leq \sum_{j=0}^{\infty} \left\| \psi_j(\mathbf{P}^T) \mathbf{D}^{-1} \mathbf{r}_j \right\|_\infty \tag{7.45}$$

$$\leq \sum_{j=0}^{\infty} \psi_j(\|\mathbf{P}^T\|_\infty) \|\mathbf{D}^{-1} \mathbf{r}_j\|_\infty \tag{7.46}$$

$$= \sum_{j=0}^{\infty} \psi_j(1) \|\mathbf{D}^{-1} \mathbf{r}_j\|_\infty, \tag{7.47}$$

where the last step follows because $\mathbf{P}$ is column-stochastic, so $\mathbf{P}^T$ is row-stochastic. We will just abbreviate $\psi_j = \psi_j(1)$. Recall that the goal is to determine a sequence of values δ_j such that running the Push method until $\|\mathbf{D}^{-1}\mathbf{r}_j\|_\infty \leq \delta_j$ will guarantee $\|\mathbf{D}^{-1}(\mathbf{f} - \hat{\mathbf{f}})\|_\infty \leq \varepsilon$. Following our noses, we will look at the choice

$$\|\mathbf{D}^{-1}\mathbf{r}_j\|_\infty \leq \delta_j = \tilde{\varepsilon}/\psi_j, \tag{7.48}$$

where $\tilde{\varepsilon}$ represents a scaled version of ε we will determine shortly. Before proceeding, we will show that only a finite number of the residual blocks will have nonzero norm. Assume for the moment without justification that $\|\mathbf{r}_j\|_1 \leq 1$ for all j, throughout the algorithm. Then $\|\mathbf{D}^{-1}\mathbf{r}_j\|_\infty \leq \|\mathbf{r}_j\|_1 \leq 1$. If the right-hand side in (7.48) is greater than 1, then the residual block $\mathbf{r}_j$ would *always* satisfy the convergence criterion (7.48) automatically, i.e. without ever performing any push operations on that residual block. This occurs when $1 \leq \tilde{\varepsilon}/\psi_j$, which occurs precisely when $\psi_j \leq \tilde{\varepsilon}$. But note that $\psi_j = \sum_{k=j}^{\infty} c_k$ is exactly the error from approximating the diffusion with just the first j terms, $c_0, \cdots, c_{j-1}$. Hence, if we determine N such that $|\psi - \sum_{k=0}^{N} c_k| < \tilde{\varepsilon}$, then we have $\psi_N \leq \tilde{\varepsilon}$ which implies the residual block $\mathbf{r}_j$ automatically satisfies the criterion $\|\mathbf{D}^{-1}\mathbf{r}_j\|_\infty \leq \delta_j$ for all $j \geq N$ (which means no pushes occur on those blocks).

Finally we justify the claim that $\|\mathbf{r}_j\|_1 \leq 1$ for all j. Because the initial residual is $\mathbf{r}^{(0)} = \mathbf{e}_1 \otimes \mathbf{s}$, we know that $\|\mathbf{r}^{(0)}\|_1 = \|\mathbf{s}\|_1 = 1$, because the seed vector is normalized to be stochastic. Next, recall that the iterative update to the residual is $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - r\mathbf{e}_i \otimes \mathbf{e}_j + (r\mathbf{e}_{i+1} \otimes \mathbf{P}\mathbf{e}_j)$, where r is the value of entry $(\mathbf{e}_i \otimes \mathbf{e}_j)^T\mathbf{r}^{(k)}$. We claim that $\|\mathbf{r}^{(k+1)}\|_1 = \|\mathbf{r}^{(k)}\|_1$ for all k. To see this, we use the fact that the residual is nonnegative for all k, and consider that $\|\mathbf{r}^{(k+1)}\|_1 = \mathbf{e}^T\mathbf{r}^{(k+1)}$, and

$$\begin{aligned}
\mathbf{e}^T\mathbf{r}^{(k+1)} &= \mathbf{e}^T(\mathbf{r}^{(k)} - r\mathbf{e}_i \otimes \mathbf{e}_j + (r\mathbf{e}_{i+1} \otimes \mathbf{P}\mathbf{e}_j)) \\
&= \mathbf{e}^T\mathbf{r}^{(k)} - r + r\mathbf{e}^T(\mathbf{e}_{i+1} \otimes \mathbf{P}\mathbf{e}_j) \\
&= \|\mathbf{r}^{(k)}\|_1 - r + r(\mathbf{e}^T\mathbf{P}\mathbf{e}_j) \\
&= \|\mathbf{r}^{(k)}\|_1 - r + r,
\end{aligned}$$

proving the claim. Thus, $\|\mathbf{r}^{(k)}\|_1 = 1$ for all k, and since $1 = \|\mathbf{r}^{(k)}\|_1 = \sum_{j=0}^{\infty} \|\mathbf{r}_j^{(k)}\|_1$, we have proved that $\|\mathbf{r}_j^{(k)}\|_1 \leq 1$ for all k and all j.

7.4.4 Setting push-coefficients

Putting this all together, we can guarantee convergence to ε accuracy as follows. First, determine N such that $|\psi - \sum_{k=0}^{N} c_k| = \sum_{k=N+1}^{\infty} c_k \leq \varepsilon\theta$ for some fixed $\theta \in (0, 1)$ (a practical choice is $\theta = 1/2$). Then set $\delta_j = (1 - \theta)\varepsilon/(N\psi_j)$. Finally, we can bound the error as follows. Residual blocks $\mathbf{r}_j$ for

$j \geq N+1$ are zero, (since no pushes occur on blocks $j \geq N$, this means no blocks *after* those contain any nonzero entries). Hence, the error is bounded by

$$\|\mathbf{D}^{-1}(\mathbf{f} - \hat{\mathbf{f}})\|_\infty \leq \sum_{j=0}^{\infty} \psi_j \|\mathbf{D}^{-1}\mathbf{r}_j\|_\infty \tag{7.49}$$

$$= \sum_{j=0}^{N} \psi_j \|\mathbf{D}^{-1}\mathbf{r}_j\|_\infty \qquad \text{because blocks beyond } N \text{ are } 0 \tag{7.50}$$

$$\leq \sum_{j=0}^{N-1} \psi_j \delta_j + \psi_N \tag{7.51}$$

$$\leq \sum_{j=0}^{N-1} (1-\theta)\varepsilon/N + \theta\varepsilon \tag{7.52}$$

which equals ε, proving the desired bound.

7.4.5 Generalized diffusion algorithm

We now present our generalized diffusion algorithm in the same format as our heat kernel algorithm in Section 7.1.4. Because of the relationship, we call this algorithm `gen-relax`, after `hk-relax` from the previous section. Given a random walk transition matrix $\mathbf{P}$ for an undirected graph, diffusion coefficients c_k satisfying $c_k \geq 0$ and $\sum_{k=0}^{\infty} c_k = 1$, a stochastic seed vector $\mathbf{s}$, and a desired accuracy ε, we approximate the diffusion $\mathbf{f} = \sum_{k=0}^{\infty} c_k \mathbf{P}^k \mathbf{s}$ with a degree-normalized infinity norm accuracy of ε by solving the linear system from (7.9) as follows.

First, compute the parameters N, θ, and δ_j as described in the previous section. Denote the solution vector by $\mathbf{y}$ and the initial residual by $\mathbf{r}^{(0)} = \mathbf{e}_1 \otimes \mathbf{s}$. Technically $\mathbf{r}$ represents an infinite-dimensioned vector, but only $N+1$ blocks will be used, so for practical implementation purposes $\mathbf{r}$ can be stored as an $(N+1)$-dimensioned vector, or better yet a hashtable. Denote by $r(i,j)$ the entry of $\mathbf{r}$ corresponding to node i in residual block j. The idea is to iteratively remove all entries from $\mathbf{r}$ that satisfy

$$r(i,j) \geq \delta_j d(k). \tag{7.53}$$

To organize this process, we begin by placing the nonzero entries of $\mathbf{r}^{(0)}$ in a queue, $Q(\mathbf{r})$, and place updated entries of $\mathbf{r}$ into $Q(\mathbf{r})$ only if they satisfy (7.53).

The algorithm proceeds as follows.

1. At each step, pop the top entry of $Q(\mathbf{r})$, call it $r(i,j)$, and subtract that entry in $\mathbf{r}$, making $\mathbf{r}(i,j) = 0$.
2. Add $c_j \cdot r(i,j)$ to $\mathbf{y}_i$.
3. Add $r(i,j)\mathbf{P}\mathbf{e}_i$ to residual block $\mathbf{r}_{j+1}$.
4. For each entry of $\mathbf{r}_{j+1}$ that was updated, add that entry to the back of $Q(\mathbf{r})$ if it satisfies (7.10).

Once all entries of $\mathbf{r}$ that satisfy (7.53) have been removed, the resulting solution vector $\mathbf{y}$ will satisfy the accuracy requirement $\|\mathbf{D}^{-1}(\mathbf{f}-\mathbf{y})\|_\infty \leq \varepsilon$. Finally, we prove a bound on the work required to achieve this.

7.4.6 Bounding work

At last we prove that the amount of work required for push to converge is bounded by a constant in terms of the accuracy ε and the diffusion itself. We adapt techniques from Section 7.1 to complete the proof (those techniques were in turn generalizations of technique used in the proof that the Andersen-Chung-Lang pprpush algorithm for PageRank is constant time [Andersen et al., 2006a]).

Observe that every step of `gen-relax` operates on an entry of the residual violating the convergence criterion $\|\mathbf{D}^{-1}\mathbf{r}_j\|_\infty \leq \delta_j$. Each such operation is on a residual entry corresponding to a node, k, in a particular residual block $\mathbf{r}_j$; we denote such an entry by $r(j,k)$. Since this entry violates the convergence criterion, we know $r(j,k) > \delta_j d(k)$.

The two key insights here are that the sum of all residual entries in block j are bounded above by 1, and the total work performed is equal to the sum of $d(k)$ for all such residual entries that we push. This means that, letting operations $t_j = 1 : m_j$ be all of the steps that we push on an entry in block j of the residual, we know $1 \geq \sum_{t_j=1}^{m_j} r(j,k(t_j)) \geq \sum_{t_j=1}^{m_j} \delta_j d(k(t_j))$. In other words, the work performed to clear residual block j is $\sum_{t_j=1}^{m_j} d(k(t_j))$, and it is bounded above by $1/\delta_j$.

The total work performed is the sum of the work performed on each residual block, and so we have

$$\begin{aligned}
\text{work}(\varepsilon) &= \sum_{j=0}^{\infty} \left(\sum_{t_j=1}^{m_j} d(k(t_j)) \right) && (7.54)\\
&\leq \sum_{j=0}^{N-1} \left(\sum_{t_j=1}^{m_j} r(j,k(t_j))/\delta_j \right) && (7.55)\\
&\leq \sum_{j=0}^{N-1} \tfrac{1}{\delta_j} \left(\sum_{t_j=1}^{m_j} r(j,k(t_j)) \right) && (7.56)\\
&\leq \sum_{j=0}^{N-1} 1/\delta_j. && (7.57)
\end{aligned}$$

Recalling that $\delta_j = (1-\theta)\varepsilon/(N\psi_j)$, we can upperbound $1/\delta_j$ with $1/\delta_0$. This is because the quantities ψ_j monotonically decrease as j increases, and so $1 = \psi = \psi_0$ is the largest of all ψ_j. Thus, we can majorize $\text{work}(\varepsilon) \leq (\sum_{j=0}^{N-1} 1/\delta) = N^2/(\varepsilon(1-\theta))$. Since we chose $\theta = 1/2$, we can write this bound as $2N^2/\varepsilon$. This completes a proof of the final theorem in this thesis:

Theorem 7.4.1 *Let $\boldsymbol{P}$, be the random walk transition matrix for an undirected graph. Let constants c_j, ψ_j, δ_j, N, θ, ε, and $\boldsymbol{r}$ be as described in the notation of Section 7.4.5 above. If steps of* `gen-relax` *are performed until all entries of the residual satisfy $r(i,j) \leq \delta_j d(k)$, then* `gen-relax` *produces an approximation $\boldsymbol{y}$ of $\boldsymbol{f} = \sum_{k=0}^{\infty} c_k \boldsymbol{P}^k \boldsymbol{s}$ satisfying*

$$\|\boldsymbol{D}^{-1}(\boldsymbol{f} - \boldsymbol{y})\|_\infty \leq \varepsilon,$$

and the amount of work required satisfies

$$work(\varepsilon) \leq \tfrac{2N^2}{\varepsilon}.$$

We remark that if we apply this bound in particular to computing the heat kernel, then our tighter analysis in this section gives an improved bound on the amount of work required to compute the heat kernel, compared to the bound we derived in the previous section. In particular, our previous work bound for the heat kernel scales with $O(Ne^t/\varepsilon)$, whereas our result in this section gives a bound that scales with $O(N^2/\varepsilon)$. By Lemma 7.1.1 we know that N scales with t at worst at the rate $N \sim t \log t$, making $N^2 \ll Ne^t$. Thus, our general framework gives a tighter bound on the runtime of the heat kernel.

8. CONCLUSIONS AND FUTURE WORK

In this thesis we consider the problem of rapidly analyzing a graph near a target region, specifically by computing graph diffusions and matrix functions locally. We consider two categories of localization: weak and strong localization, characterized by the type of norm used to measure convergence. We study conditions on the diffusions and the graph themselves that can guarantee the presence of weak and strong localization. We summarize our findings here, and discuss potential continuations of this research.

Regarding weak localization, we prove that an entire class of graph diffusions exhibit weak localization on all undirected graphs, and propose a novel algorithm for locally computing these, in constant-time. Our empirical evaluations show that the heat kernel diffusion outperforms the standard PageRank diffusion. These results make us hopeful that our algorithm for computing general diffusions will enable a more comprehensive study of the performance of different kinds of diffusions in community detection and other applications of weakly local diffusions. We also leave as future work the task of proving a Cheeger inequality for general diffusions that bounds the best conductance obtainable from a sweep-cut over a diffusion to the coefficients of that diffusion as well as the optimal conductance attainable near the seed node. Such a Cheeger inequality could then guide a more strategic choice of diffusion coefficients, possibly optimized for locating sets of the optimal conductance.

The case of strong localization of matrix functions turns out to be more complicated than that of weak localization. It was already known in the literature that some functions exhibit localization on graphs with constant maximum-degree. We advance the literature by demonstrating that both PageRank (the resolvent function) and the matrix exponential exhibit strong localization on graphs that have a particular type of skewed degree sequence closely related to the power-law degree sequence property common to many real-world networks. Furthermore, we show that there exist categories of graphs (namely, complete-bipartite graphs) for which many functions are totally de-localized. This creates a gap: what can we say about graphs that are less dense or larger diameter than complete-bipartite graphs, but more dense or smaller diameter than the skewed degree sequence graphs that we consider? As discussed in Section 3.2, we believe at the very least that edge-perturbations of complete-bipartite graphs will also exhibit total de-localization for some functions.

Finally, we propose several algorithms for rapidly computing the matrix exponential, as well as a new framework for approximating a matrix-vector product by rounding entries in a near-optimal

way. Our first algorithm generalizes the Gauss-Southwell sparse linear solver to apply to the matrix exponential instead of solving a linear system. We call this method `gexpm`. Though we prove that `gexpm` is sublinear on graphs with a skewed degree sequence, and it runs quickly on sparser graphs, our `gexpm` algorithm for the matrix exponential slows down on larger, denser graphs (we believe because of expensive heap-updates involved in the algorithm). Because of this, we explore other routines in Chapter 5 that avoid these expensive heap updates, and proceed essentially by rounding the intermediate vectors prior to performing matrix-vector products. One of our methods, `gexpmq`, determines a rounding threshold at each step to guarantee convergence to the desired accuracy; although this method outperforms `gexpm` in practice, it remains to be proved that `gexpmq` has a sublinear work bound. We suspect this can be accomplished by leveraging additional assumptions on the connectivity structure of the underlying graph. The other rounding method, `expmimv`, rounds to zero all but a constant number of entries to guarantee a sublinear work bound, but at the cost of rigorous control on the accuracy of its output.

Because of the empirical success of the above methods, we also study the problem of how to optimally determine entries in a vector to round to zero so as to minimize the amount of work required in computing a rounded-matrix vector product. We propose a new approximation algorithm for the Knapsack Problem and show that our routine is linear time, improving on a standard greedy approximation algorithm for the Knapsack Problem, and we show this enables near-optimal selection of which entries to round to zero to perform a rounded matrix-vector product. We envision this fast rounding procedure being used to perform repeated rounded matrix-vector products with minimal fill-in, so as to rapidly compute a polynomial of a matrix times a vector. This has applications to approximating matrix functions, solving linear systems, and even eigensystems. We are hopeful that careful analysis of the fill-in occurring during the rounded matrix-vector products could yield another sublinear work bound in computing a broader class of functions of matrices.

The work in this thesis demonstrates that a variety of diffusions can be computed efficiently for applications requiring both strong and weak convergence. Our generalized diffusion framework enables the rapid, weakly local computation of any diffusion that decays quickly, regardless of graph structure. This means that such weakly local graph diffusions are computable in constant time for any graph, and suggests that other weakly local graph computations might also be possible in constant or sublinear time. On the other hand, the results in this thesis also demonstrate that there are still a number of open questions on strong localization in graph diffusions. We identify a regime of graph structures for which strong localization behavior is unknown – graphs less connected than complete-bipartite graphs but more connected than graphs with our skewed degree sequence. Furthermore, our results show only that PageRank and the matrix exponential are localized on these

skewed degree sequence graphs, leaving as another open question which other such matrix functions exhibit such strong localization and which do not.

It is interesting to note that all of localization results, both weak and strong, follow from analyzing the convergence of algorithms. This suggests that future algorithmic developments could yield improved bounds on localization and raises the question of whether a non-constructive analytic approach might yield even tighter bounds. Finally, this thesis focuses on deterministic algorithms for computing diffusions and does not consider Monte Carlo approaches. Much recent work has shown that Monte Carlo methods for diffusions can rapidly produce rough approximations but slow down significantly to obtain higher accuracy approximations. Unifying the deterministic and Monte Carlo approaches is an interesting for future research. For instance, although our deterministic methods are generally fast, they can be slowed down by nodes of very large degree, but a hybrid approach to computing a graph diffusion could circumvent this difficulty by employing Monte Carlo subroutines to handle problematic graph structures like high degree nodes.

REFERENCES

REFERENCES

Lada A. Adamic. Zipf, power-laws, and pareto – a ranking tutorial, 2002. URL `http://www.hpl.hp.com/research/idl/papers/ranking/ranking.html`. Accessed on 2014-09-08.

Awad H. Al-Mohy and Nicholas J. Higham. Computing the action of the matrix exponential, with an application to exponential integrators. *SIAM J. Sci. Comput.*, 33(2):488–511, 2011. ISSN 1064-8275. doi: 10.1137/100788860.

R. Alberich, J. Miro-Julia, and F. Rossello. Marvel universe looks almost like a real social network. *arXiv*, cond-mat.dis-nn:0202174, 2002. URL `http://arxiv.org/abs/cond-mat/0202174`.

Reid Andersen and Kevin J. Lang. Communities from seed sets. In *Proceedings of the 15th international conference on the World Wide Web*, pages 223–232, New York, NY, USA, 2006. ACM Press. doi: 10.1145/1135777.1135814.

Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using PageRank vectors. In *FOCS2006*, 2006a.

Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using PageRank vectors. `http://www.math.ucsd.edu/~randerse/papers/local_partitioning_full.pdf`, 2006b. URL `http://www.math.ucsd.edu/~randerse/papers/local_partitioning_full.pdf`. Extended version of [Andersen et al., 2006a].

Konstantin Avrachenkov, Nelly Litvak, Marina Sokol, and Don Towsley. Quick detection of nodes with large degrees. In Anthony Bonato and Jeannette Janssen, editors, *Algorithms and Models for the Web Graph*, volume 7323 of *Lecture Notes in Computer Science*, pages 54–65. Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-30541-2_5.

Haim Avron and Lior Horesh. Community detection using time-dependent personalized pagerank. In *ICML*, pages 1795–1803, 2015.

Ricardo Baeza-Yates, Paolo Boldi, and Carlos Castillo. Generalizing PageRank: Damping functions for link-based ranking algorithms. In *SIGIR2006*, pages 308–315, 2006.

M Benzi and N Razouk. Decay bounds and O(n) algorithms for approximating functions of sparse matrices. *ETNA*, 28:16–39, 2007.

Michele Benzi and Christine Klymko. Total communicability as a centrality measure. *Journal of Complex Networks*, 1(2):124–149, 2013.

Michele Benzi, Paola Boito, and Nader Razouk. Decay properties of spectral projectors with applications to electronic structure. *SIAM Review*, 55(1):3–64, 2013.

Pavel Berkhin. Bookmark-coloring algorithm for personalized PageRank computing. *Internet Mathematics*, 3(1):41–62, 2007.

Marián Boguñá, Romualdo Pastor-Satorras, Albert Díaz-Guilera, and Alex Arenas. Models of social networks based on social distance attachment. *Phys. Rev. E*, 70(5):056122, 2004. doi: 10.1103/PhysRevE.70.056122.

Paolo Boldi and Sebastiano Vigna. Codes for the world wide web. *Internet Mathematics*, 2(4): 407–429, 2005. URL `http://www.internetmathematics.org/volumes/2/4/Vigna.pdf`.

Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *Proceedings of the 20th WWW2011*, pages 587–596, March 2011. doi: 10.1145/1963405.1963488.

Francesco Bonchi, Pooya Esfandiar, David F. Gleich, Chen Greif, and Laks V.S. Lakshmanan. Fast matrix computations for pairwise and columnwise commute times and Katz scores. *Internet Mathematics*, 8(1-2):73–112, 2012.

Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, Michael Mitzenmacher, Alessandro Panconesi, and Prabhakar Raghavan. On compressing social networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 219–228, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-495-9. doi: 10.1145/1557019.1557049. URL `http://doi.acm.org/10.1145/1557019.1557049`.

Fan Chung. The heat kernel as the PageRank of a graph. *Proceedings of the National Academy of Sciences*, 104(50):19735–19740, December 2007a.

Fan Chung. The heat kernel as the PageRank of a graph. *Proceedings of the National Academy of Sciences*, 104(50):19735–19740, December 2007b.

Fan Chung. A local graph partitioning algorithm using heat kernel pagerank. *Internet Mathematics*, 6(3):315–330, 2009.

Fan Chung and Olivia Simpson. Solving linear systems with boundary conditions using heat kernel pagerank. In *Algorithms and Models for the Web Graph*, pages 203–219. Springer, 2013.

Fan RK Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.

George B. Dantzig. Discrete-variable extremum problems. *Operations Research*, 5(2):266–288, 1957. doi: 10.1287/opre.5.2.266. URL `http://dx.doi.org/10.1287/opre.5.2.266`.

Ernesto Estrada. Characterization of 3d molecular structure. *Chemical Physics Letters*, 319(5-6): 713–718, 2000. ISSN 0009-2614. doi: 10.1016/S0009-2614(00)00158-5.

Ernesto Estrada and Desmond J. Higham. Network properties revealed through matrix functions. *SIAM Review*, 52(4):696–714, 2010. doi: 10.1137/090761070.

Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. *SIGCOMM Comput. Commun. Rev.*, 29:251–262, August 1999a. ISSN 0146-4833. doi: 10.1145/316194.316229.

Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *ACM SIGCOMM computer communication review*, 1999b.

Ayman Farahat, Thomas LoFaro, Joel C. Miller, Gregory Rae, and Lesley A. Ward. Authority rankings from HITS, PageRank, and SALSA: Existence, uniqueness, and effect of initialization. *SIAM Journal on Scientific Computing*, 27(4):1181–1201, 2006. doi: 10.1137/S1064827502412875.

Valerio Freschi. Protein function prediction from interaction networks using a random walk ranking algorithm. In *BIBE*, pages 42–48, 2007.

Gérard Meurant Gene H. Golub. *Matrices, Moments and Quadrature with Applications*. Princeton University Press, 2010. ISBN 9780691143415. URL `http://www.jstor.org/stable/j.ctt7tbvs`.

Rumi Ghosh, Shang-hua Teng, Kristina Lerman, and Xiaoran Yan. The interplay between dynamics and networks: Centrality, communities, and cheeger inequality. pages 1406–1415, 2014.

David F. Gleich. PageRank beyond the web. *SIAM Review*, 57(3):321–363, August 2015a. doi: 10.1137/140976649.

Kyle Gleich, David F. and Kloster. Sublinear column-wise actions of the matrix exponential on social networks. *Internet Mathematics*, 11(4-5):352–384, 2015b.

Marco Gori and Augusto Pucci. ItemRank: a random-walk based scoring algorithm for recommender engines. In *IJCAI*, pages 2766–2771, 2007.

Nicholas J. Higham. *Functions of Matrices: Theory and Computation*. SIAM, 2008.

Jun Hirai, Sriram Raghavan, Hector Garcia-Molina, and Andreas Paepcke. Webbase: a repository of web pages. *Computer Networks*, 33(1-6):277–293, June 2000. doi: 10.1016/S1389-1286(00)00063-3.

Bernardo A. Huberman, Peter L. T. Pirolli, James E. Pitkow, and Rajan M. Lukose. Strong regularities in World Wide Web surfing. *Science*, 280(5360):95–97, 1998.

Alpa Jain and Patrick Pantel. Factrank: Random walks on a web of facts. In *COLING*, pages 501–509, 2010.

G. Jeh and J. Widom. Scaling personalized web search. In *WWW*, pages 271–279, 2003.

Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, March 1953. doi: 10.1007/BF02289026.

Kyle Kloster and David F. Gleich. *Algorithms and Models for the Web Graph: 10th International Workshop, WAW 2013, Cambridge, MA, USA, December 14-15, 2013, Proceedings*, chapter A Nearly-Sublinear Method for Approximating a Column of the Matrix Exponential for Matrices from Large, Sparse Networks, pages 68–79. Springer International Publishing, Cham, 2013.

Kyle Kloster and David F Gleich. Heat kernel based community detection. In *KDD*, pages 1386–1395, 2014.

Risi Imre Kondor and John D. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *ICML '02*, pages 315–322, 2002. ISBN 1-55860-873-7.

Jérôme Kunegis and Andreas Lommatzsch. Learning spectral graph transformations for link prediction. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 561–568, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553447.

Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is Twitter, a social network or a news media? In *WWW '10: Proceedings of the 19th international conference on World wide web*, pages 591–600, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-799-8. doi: 10.1145/1772690.1772751.

Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data*, 1:1–41, March 2007. ISSN 1556-4681. doi: 10.1145/1217299.1217301.

Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, September 2009. doi: 10.1080/15427951.2009.10129177.

Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Signed networks in social media. In *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, pages 1361–1370, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-929-9. doi: 10.1145/1753326.1753532.

Yixuan Li, Kun He, David Bindel, and John Hopcroft. Uncovering the small community structure in large networks: A local spectral approach. *WWW*, 2015.

ML Liou. A novel method of evaluating transient response. *Proceedings of the IEEE*, 54(1):20–23, 1966.

Z. Q. Luo and P. Tseng. On the convergence of the coordinate descent method for convex differentiable minimization. *J. Optim. Theory Appl.*, 72(1):7–35, 1992. ISSN 0022-3239. doi: 10.1007/BF00939948.

Frank McSherry. A uniform approach to accelerated PageRank computation. In *Proceedings of the 14th international conference on the World Wide Web*, pages 575–582, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-046-9. doi: 10.1145/1060745.1060829.

Borislav V. Minchev and Will M. Wright. A review of exponential integrators for first order semi-linear problems. Technical Report Numerics 2/2005, Norges Teknisk-Naturvitenskapelige Universitet, 2005. URL `http://www.ii.uib.no/~borko/pub/N2-2005.pdf`.

Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *SIGCOMM*, pages 29–42, 2007. ISBN 978-1-59593-908-1. doi: 10.1145/1298306.1298311.

C. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Review*, 20(4):801–836, 1978. ISSN 00361445.

C. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45(1):3–49, 2003. doi: 10.1137/S00361445024180.

Julie L Morrison, Rainer Breitling, Desmond J Higham, and David R Gilbert. Generank: using search engine technology for the analysis of microarray experiments. *BMC bioinformatics*, 6(1):233, 2005.

Huda Nassar, Kyle Kloster, and David F Gleich. Strong localization in personalized pagerank vectors. In *Algorithms and Models for the Web Graph*, pages 190–202. Springer International Publishing, 2015.

M. E. J. Newman. The structure of scientific collaboration networks. *Proceedings of the National Academy of Sciences*, 98(2):404–409, 2001. doi: 10.1073/pnas.98.2.404.

Mark Newman. Network datasets. `http://www-personal.umich.edu/~mejn/netdata/`, 2006.

Zaiqing Nie, Yuanzhi Zhang, Ji-Rong Wen, and Wei-Ying Ma. Object-level ranking: Bringing order to web objects. In *WWW*, pages 567–574, 2005.

Lorenzo Orecchia and Michael W. Mahoney. Implementing regularization implicitly via approximate eigenvector computation. In *ICML*, pages 121–128, 2011. URL `http://www.icml-2011.org/papers/120_icmlpaper.pdf`.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford University, 1999.

Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007. ISSN 1574-0137. doi: 10.1016/j.cosrev.2007.05.001.

Xin Sui, Tsung-Hsien Lee, Joyce Jiyoung Whang, Berkant Savas, Saral Jain, Keshav Pingali, and Inderjit Dhillon. Parallel clustered low-rank approximation of graphs and its application to link prediction. In *Languages and Compilers for Parallel Computing*, volume 7760 of *Lecture Notes in Computer Science*, pages 76–95. Springer Berlin, 2013. ISBN 978-3-642-37657-3. doi: 10.1007/978-3-642-37658-0_6.

CAIDA (The Cooperative Association for Internet Data Analyais). Network datasets. `http://www.caida.org/tools/measurement/skitter/router_topology/`, 2005. Accessed in 2005.

Jaewon Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 745–754, Dec 2012. doi: 10.1109/ICDM.2012.138.

Xiao-Tong Yuan and Tong Zhang. Truncated power method for sparse eigenvalue problems. *CoRR*, abs/1112.2679, 2011.

VITA

VITA

Kyle Kloster was born in St. Louis, Missouri in 1987. In May 2010 he received a B.S. in Mathematics from Fordham University where he worked part time as a tutor in the Mathematics department, in data-entry in the Psychology department's rat laboratory, and as an archivist in the university's press. He then enrolled in Purdue University's Mathematics graduate program where he went on to receive the department's Excellence in Teaching Award as a graduate teaching assistant, as well as numerous Officemate of the Week Awards. While at Purdue he co-organized the Mathematics department's Student Colloquium, and founded and organized the inter-disciplinary Student Numerical Linear Algebra Seminar.